ZINC Tutorial Manual

Introduction 1
Example 1: Dielectric Problem 2
Introduction 2
Geometry and meshing 5
Input file *.zin 13
Running the simulation 14
Viewing the output. 16
A quick look at the result 17
Post processing 18
Traceability 24
More advanced graph plotting 24
If you don't like gnuplot 31
Also in the directory 31
Example 2: piezoelectric example 31
Introduction 31
Simulation 34
Results 37
Comparison graphs 42
Also in the directory 43
Example 3: Fuel cell 43
Introduction 43
Zinc implementation 48
DLL method 49
Expressions method 53
Running the simulation 54
Also in the directory 60
Appendix A: Description of differential equations 60

Introduction

This tutorial manual shows a series of real life problems being solved with Zinc. It is probably the best place to learn how to use the program, though it may be helpful to refer back to the User Manual from time to time. Each tutorial corresponds to one of the examples in the "examples" folder that came with your Zinc distribution. These folders contain the input and output files for the simulation as generated by us. When you run the simulations, you should check you get the same answers we did. It's probably a good idea to copy the input files to a fresh folder so as not the overwrite the output files we've supplied. That way, you can do a comparison easily.

Example 1: Dielectric Problem

Directory: examples/dielectric

Introduction

ZINC is a finite element program that is designed to solve problems having the form

$$-\nabla.\left(\mathbf{C}\nabla\mathbf{U}\right)+\mathbf{a}\,\mathbf{U}=\mathbf{f}\,, \tag{1}$$

which is written explicitly in component form as follows

$$-\frac{\partial}{\partial x_i}\left(\overline{C}_{ij}^{\alpha\beta}\,\frac{\partial U^{\beta}}{\partial x_j}\right)+a^{\alpha\beta}U^{\beta}=f^{\alpha},\quad \alpha=1,2,...,N \tag{2}$$

where summation over 1, 2, …. N (where N is the number of variables) is implied by repeated *Greek superscripts*, and where summation over $1, 2, 3$ is implied by *Arabic suffices*. Appendix A gives a more detailed description of the notation and definitions. The reader can also refer to the Zinc User Manual, Chapter 4, for a more detailed description.

In this tutorial, the problem is to determine the electrical potential distribution, and other derived properties such as the electric field and displacement field. The system consists of a spherical sample of one type of material that is embedded in a cuboid of another material. The system is excited by a potential difference between opposing faces of the cuboid. This problem is in fact mathematically equivalent to both thermal and electrical conduction problems, and to diffusion and magnetic permeability problems.

The field equations that must be solved is one of Maxwell's equations, in the absence of distributed charge, namely,

$$\nabla.\mathbf{D}=0\,, \tag{3}$$

where **D** is the electric displacement whose value is given by the constitutive relation

$$\mathbf{D}=\varepsilon\varepsilon_0\mathbf{E}\,, \tag{4}$$

where **E** is the electric field, ε_0 is the permittivity of free space and ε is the relative permittivity (a dimensionless material property). The electric field is related to the electric potential V according to the relation

$$\mathbf{E}=-\nabla\mathrm{V}\,. \tag{5}$$

The relations (3)-(5) imply that the electric potential must satisfy the equation

$$-\nabla.\left(\varepsilon\varepsilon_0\nabla\mathrm{V}\right)=0\,. \tag{6}$$

In component form this equation is written

$$-\left(\varepsilon\varepsilon_0 V_{,j}\right)_{,j} = 0\,. \tag{7}$$

The relation (7) results from (2) if the following identifications are made (See Zinc User Manual, Chapter 4)

$$N = 1, \quad a^{11} = 0, \quad f^1 = 0, \quad U_1 = V, \quad \bar{C}_{ij}^{11} = \varepsilon\varepsilon_0\delta_{ij}, \tag{8}$$

where δ_{ij} is the Kronecker delta symbol having the value 1 if i = j, 0 otherwise. It follows from (A10) that the C matrix used by ZINC has the values given by

$$\bar{C}_{1i1j} = \bar{C}_{ij}^{11} = \varepsilon\varepsilon_0\delta_{ij} \tag{9}$$

It should be noted that in ZINC all coefficients appearing in the general differential equations (2) are zero unless non-zero values are set in the ZINC input files *.zin where * denotes an input filename introduced by the user.

The differential equation (7) has a unique solution only if suitable boundary conditions are applied.

The boundary condition assumed by Zinc is

$$n_i \bar{C}_{ij}^{\alpha\beta} \frac{\partial U^\beta}{\partial x_j} + q^{\alpha\beta} U = g^\alpha$$

where **n** is the unit normal on the surface. It is easy to check that with

$$q^{11} = g^1 = 0$$

this becomes

$$\mathbf{n}\cdot\mathbf{D} = 0$$

or

$$\frac{\partial V}{\partial n} = 0$$

which is the usual open boundary condition used in electrical problems. It implies the electric field, **E**, will be parallel to the outer edges of the simulation. In the example to be solved this boundary condition applies everywhere on the external surface of the system except on two opposing planes of the cuboid where differing uniform electric potentials are to be imposed. These Dirichlet boundary conditions are imposed simply be fixed the values of the nodes at the appropriate potential (which will be simply 1 V and 0V). In ZINC, any unset components of the **C, a, f, q, g** matrices defaults to zero so the condition $\mathbf{n}\cdot\mathbf{D} = 0$ occurs automatically.

At the surface of the spherical region the normal component of the electric displacement must be continuous across the boundary, and the electric potential must also be continuous across this boundary. Provided we leave q and g at their default zero values, this condition will also occur automatically. The boundary conditions that apply are illustrated in Fig.1.

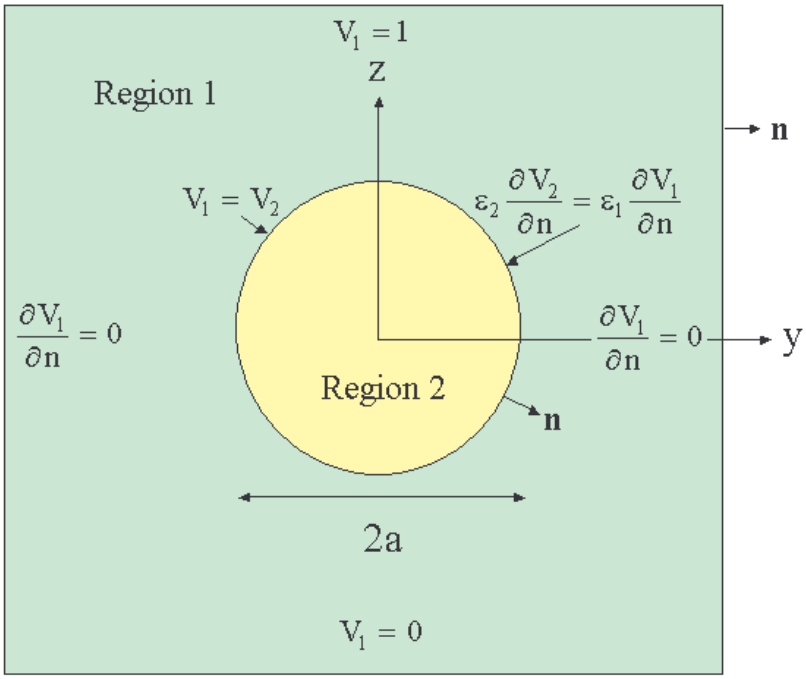


Fig.1: Schematic diagram of geometry and the boundary conditions.

The default condition on the external boundary of Region 1 is, therefore,

$$\frac{\partial V_1}{\partial n} = 0, \qquad \text{implying that} \qquad \mathbf{n}.\mathbf{D} = 0, \tag{10}$$

where **n** is the unit outward normal to the free surface.

On the plane z = 1

$$V_1 = 1, \tag{11}$$

On the plane z = −1

$$V_1 = 0. \tag{12}$$

On the interface between Regions 1 and 2 the following conditions must be satisfied

$$V_1 = V_2, \quad \varepsilon_1 \frac{\partial V_1}{\partial n} = \varepsilon_2 \frac{\partial V_2}{\partial n}, \qquad \text{implying that } \mathbf{n}.\mathbf{D} \text{ is continuous across the interface,} \tag{13}$$

where **n** is now the unit outward normal to the interface, $\partial / \partial n$ denotes the normal derivative and where ε_i is the relative permittivity of Region i. The conditions (13) ensure that the normal component of the electric displacement vector and the tangential component of the electric potential are continuous across the surface. These "continuity conditions" are true, by default, for all such internal interfaces and normally the user only needs to think about applying boundary conditions at the outer surfaces.

The other boundary condition is the Dirichlet boundary in which the unknown variables, U, are set at fixed values. In this tutorial, we will use the Dirichlet boundary at the top and bottom faces of Fig 1 and the Neumann condition (**n.D**=0) at all other faces.

Geometry and meshing

Having installed ZINC onto your computer, it is useful to generate a new folder having name 'ZINC_Runs' (or similar) into which all simulations will be generated and saved. To deal with the first example of this tutorial a new folder within 'ZINC_Runs' is created having the filename 'Dielectric'. This file will now be denoted in the following text by '*'.

Input file *.min

To represent the geometry shown in Fig.1 by finite elements the following input file must be written and stored in ZINC_Runs/Dielectric. The first step defines the global mesh of cubic elements which represents the entire region of the problem. The remainder of the file defines 'parts' of the system that are needed to define homogeneous regions having specific properties and boundary and interface conditions. In the following code comments are included to provide some explanation. More details of the *.min file are given in the Zinc User Manual.

```
global
format ascii

xmesh
  -1 1 0.2                The x range -1 < x < 1 is divided into elements of length 0.2
end

ymesh
  -1 1 0.2                The y range -1 < x < 1 is divided into elements of length 0.2
end

zmesh
  -1 1 0.2                The z range -1 < x < 1 is divided into elements of length 0.2
end

end

part 1                          Defines a cubic box having sides of length 2 as Region 1
  region 1
  type box
  fab 2 2 2
end

part 2     Defines a sphere having radius 0.5 as Region 2, inserted into Region 1
  region 2
  type sphere
  fab 0.5
  surface region 1
end

part 3                                  Defines Region 3 to be the upper plane (z = 1)
```

```
  region 3
  type boundzup
end

part 4                                  Defines Region 4 to be the lower plane (z = −1)
  region 4
  type boundzdn
end

endfile
```

The next step is to generate a mesh using zmesh.exe which appears as two windows shown in Figs. 2 and 3. The ZMesh window (Fig.2) enables the user to generate the required mesh while the Output window (Fig.3) provides information on locations of files being used.

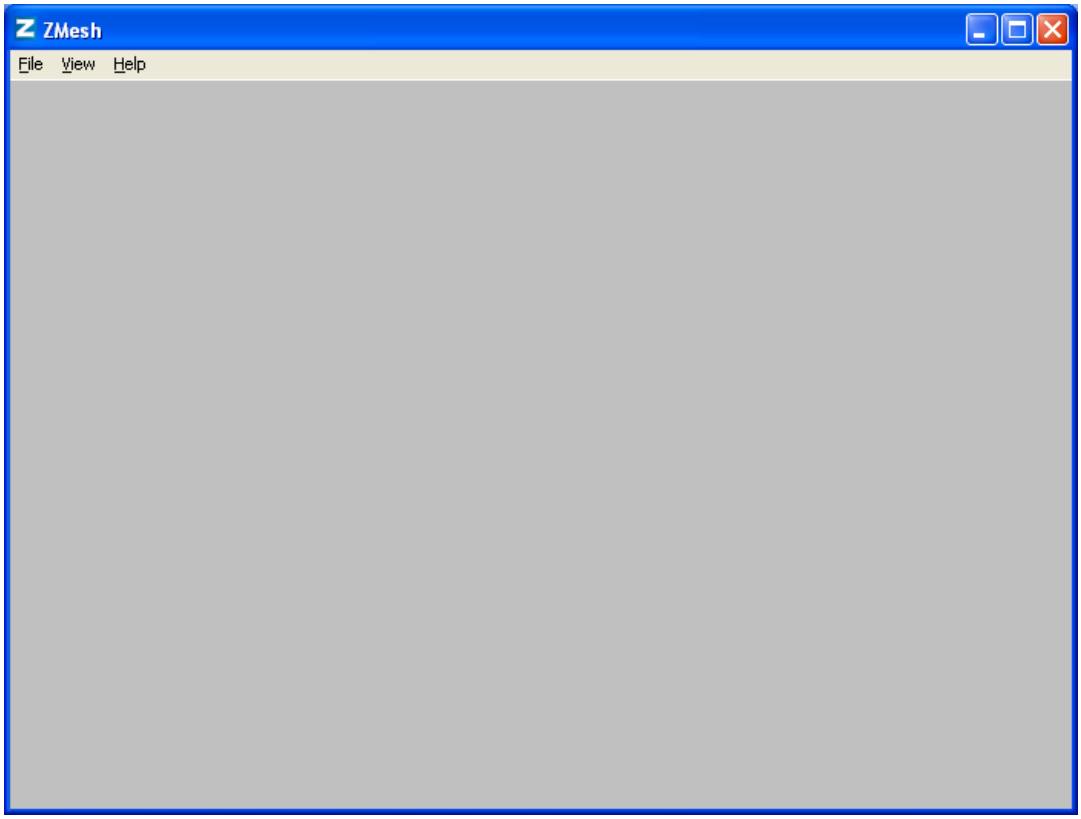


Fig.2: Window first opened by Zmesh.

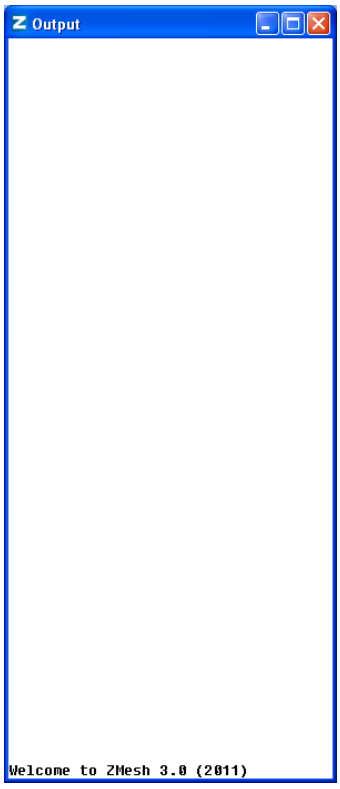


Fig.3: Output window first opened by Zmesh.

Select File > Open MIN and open the file *.min whose contents are then displayed (see Fig.4).

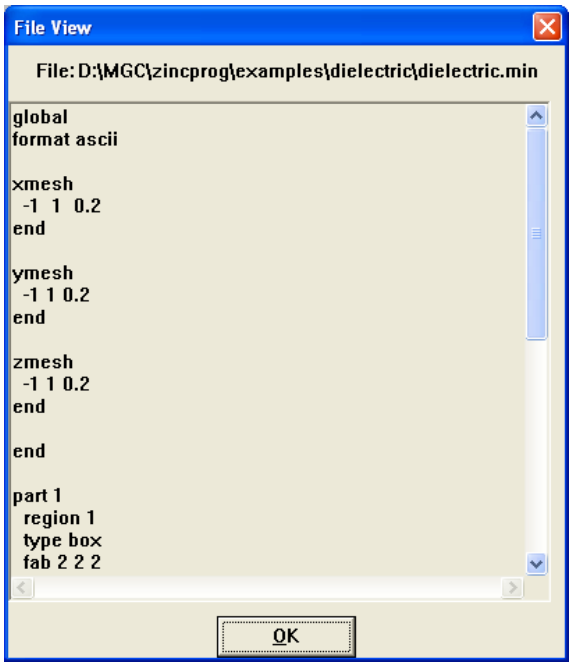


Fig.4: Window opened by Select File > Open MIN showing file *.min.

Select File/Process to generate the mesh. Provided no error occurred, the resulting mesh is shown on screen and the mesh file is automatically stored as *.mtf in the folder '*'. The main window now shows the geometry and mesh that has been generated (Fig 5). To view the geometry from different angles, drag with the mouse.

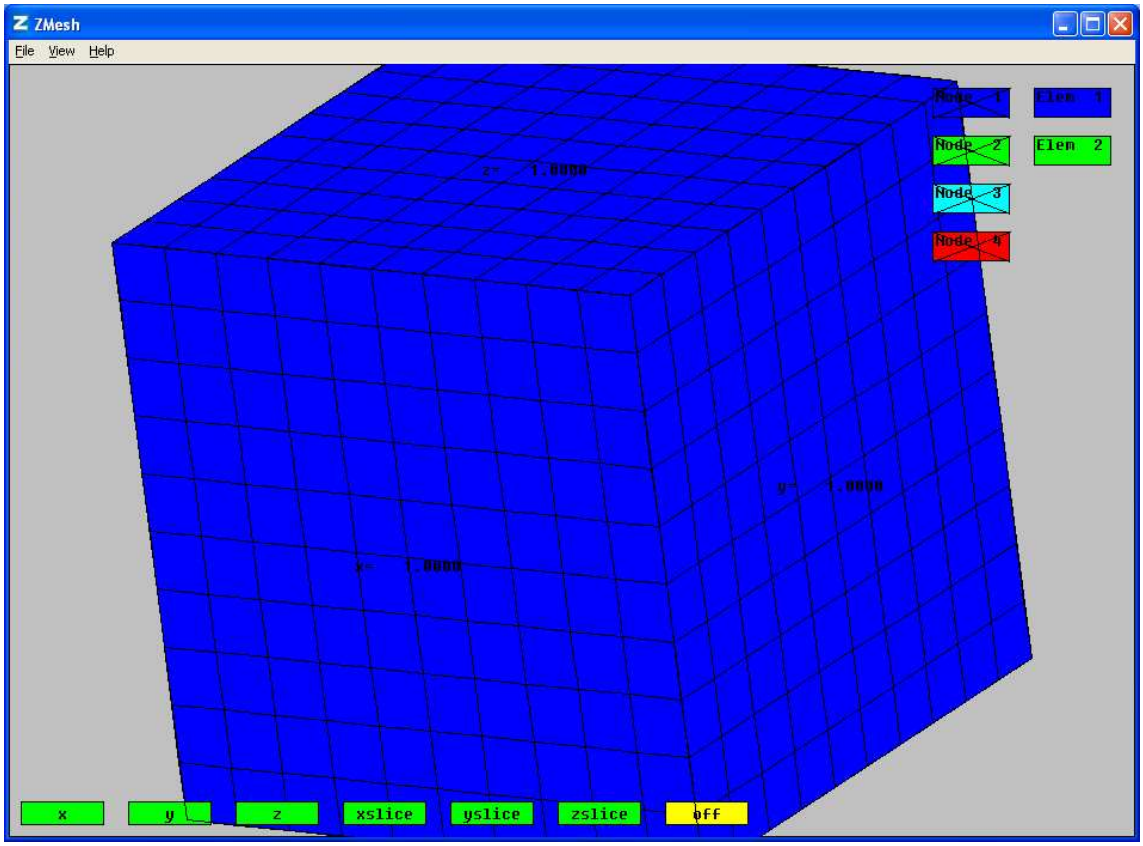


Fig.5: Window opened by Select File > Open MIN showing file *.min after rotation of model.

Dragging rotates the model so that its 3D nature can be observed. It can be returned to a standard state simply by pressing the buttons x, y or z which shows the view along the selected axis.

NOTE:
Can rotate by dragging with mouse, or use arrow keys. Hold down SHIFT while dragging with mouse to shift (rather than rotate) the figure. Press keys A/Z to zoom in/out and N/M to rotate in the plane.

To observe the sphere alone press the button 'Elem 1' (in the key at the right) which switches off the elements in Region 1 and reveals the embedded sphere in Region 2 (see Fig.6). When the elements of a region are switched off a cross appears in the key symbol. Click on the key symbol again to toggle the elements back on.

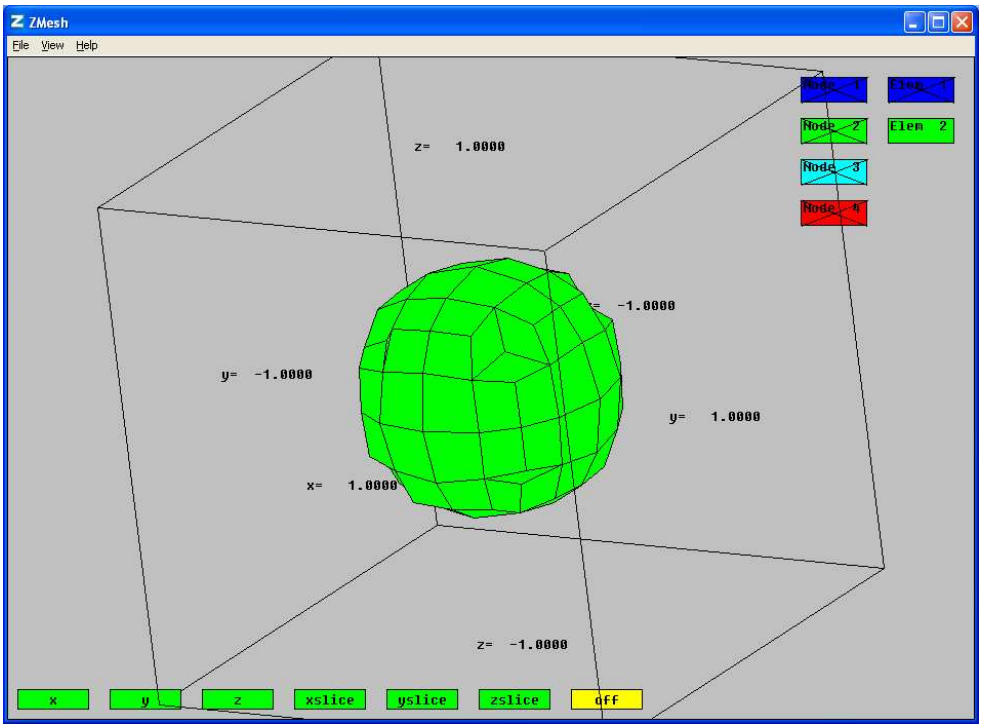


Fig.6: Spherical region is exposed by switching off the elements of Region 1.

The elements of Region 2 can be switched off by pressing the button 'Elem 2' and so on. Nodes can also be toggled on or off by clicking on the key symbols marked 'node 1' etc. The geometry of the nodes is best seen by dragging. Repeated pressing of all buttons simply toggles between two possible states; visible or invisible. In Fig.7 is shown the result of retaining the elements of Region 2 but showing the nodes.

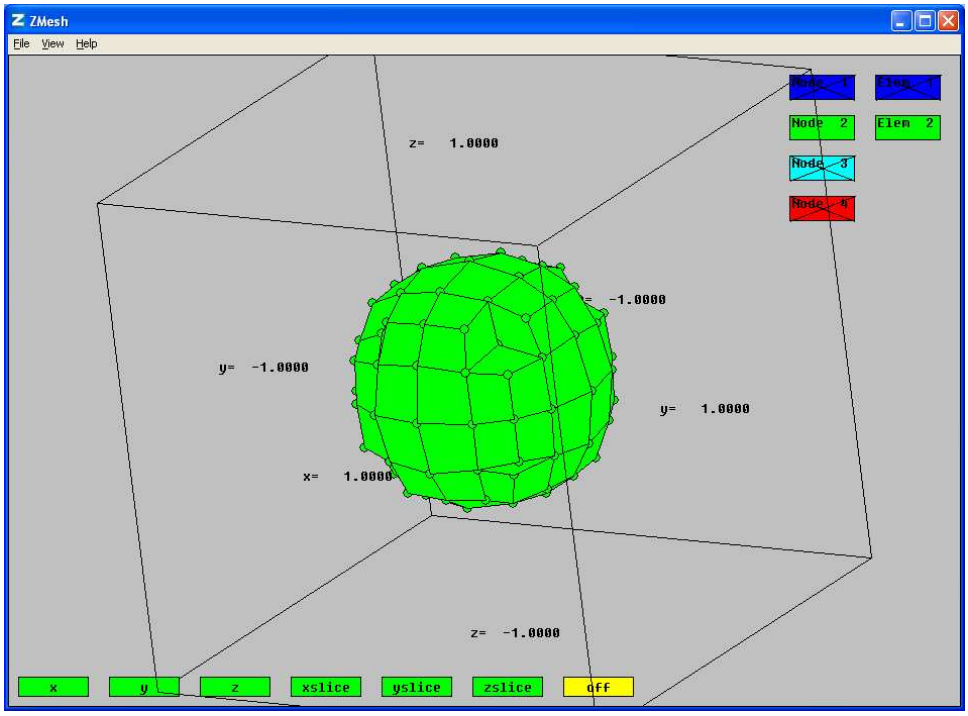


Fig.7: Elements and nodes of spherical region shown by a suitable choice of active buttons.

The colours of the nodes and elements can be changed by right-clicking on node or element buttons on top right hand side of the window. The colour is selected by specifying the mix of primary colours using the Set Colour dialogue box (see Fig.8). The result of changing node colours is shown in Fig. 9 while the result of changing element colours is shown in Fig.10.

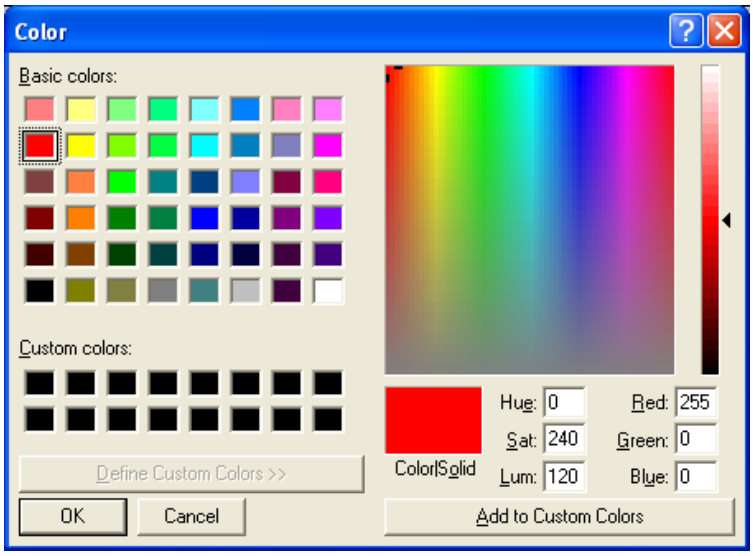


Fig.8: Dialogue box for selecting the colours of elements and nodes.

Pressing the 'xslice', 'yslice' or 'zslice' buttons shows only a slice of the model in the x, y, or z directions. The location of the slice can be changed using the Up/Down arrows on the keyboard. While doing this, you can continue to rotate the system by dragging or use keys A/Z or N/M normally. An example is shown in Fig.11 where the slice intersects both Regions 1 and 2. The elements for Region 2 can be switched off in a slice as shown in Fig.12.

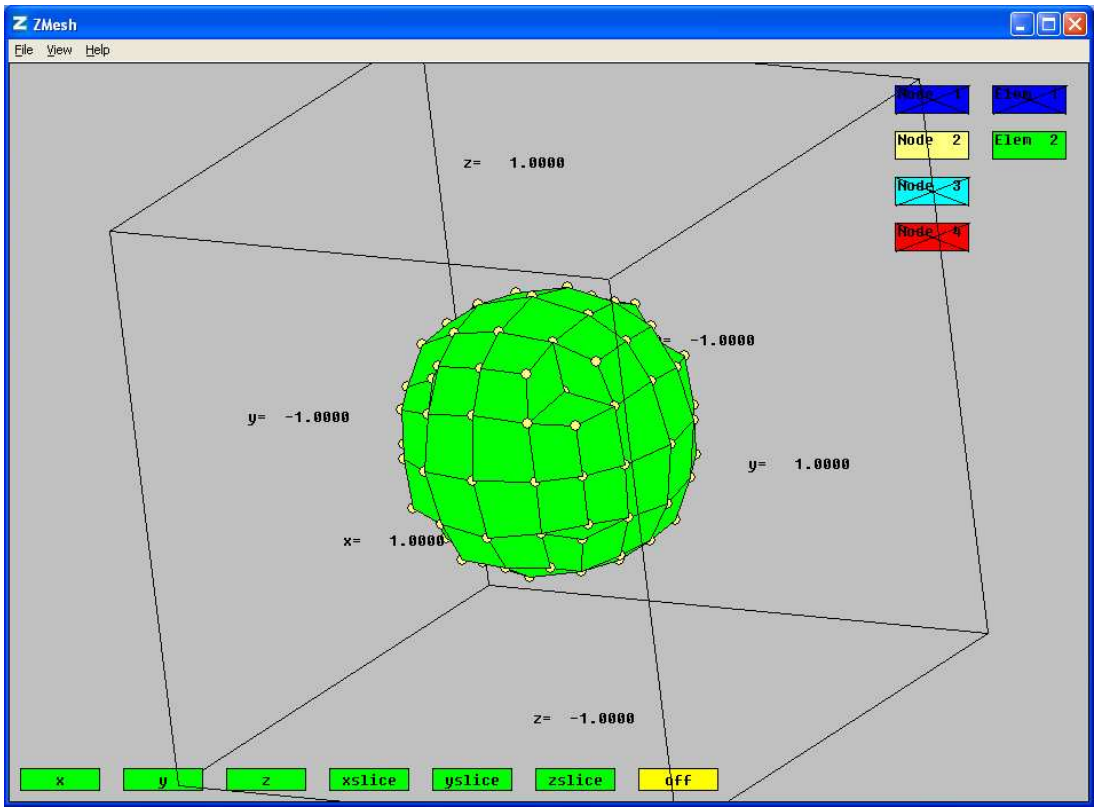


Fig.9: The result of changing the colours of the nodes of the spherical region.

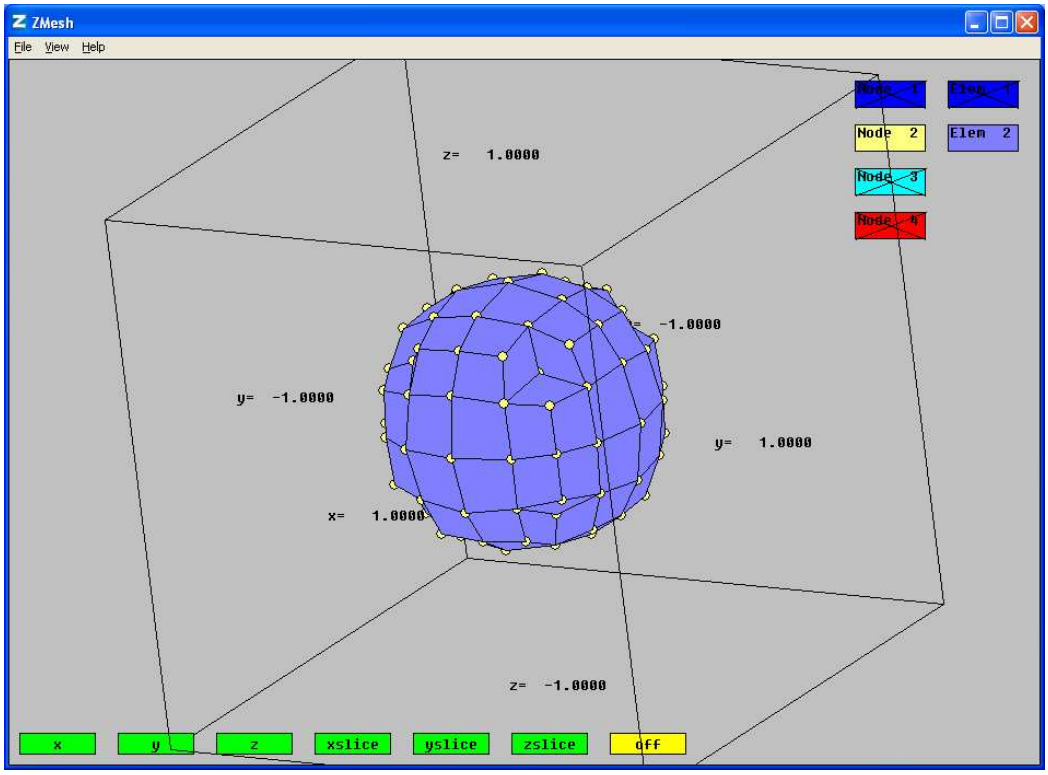


Fig.10: The result of changing the colours of the elements of the spherical region.

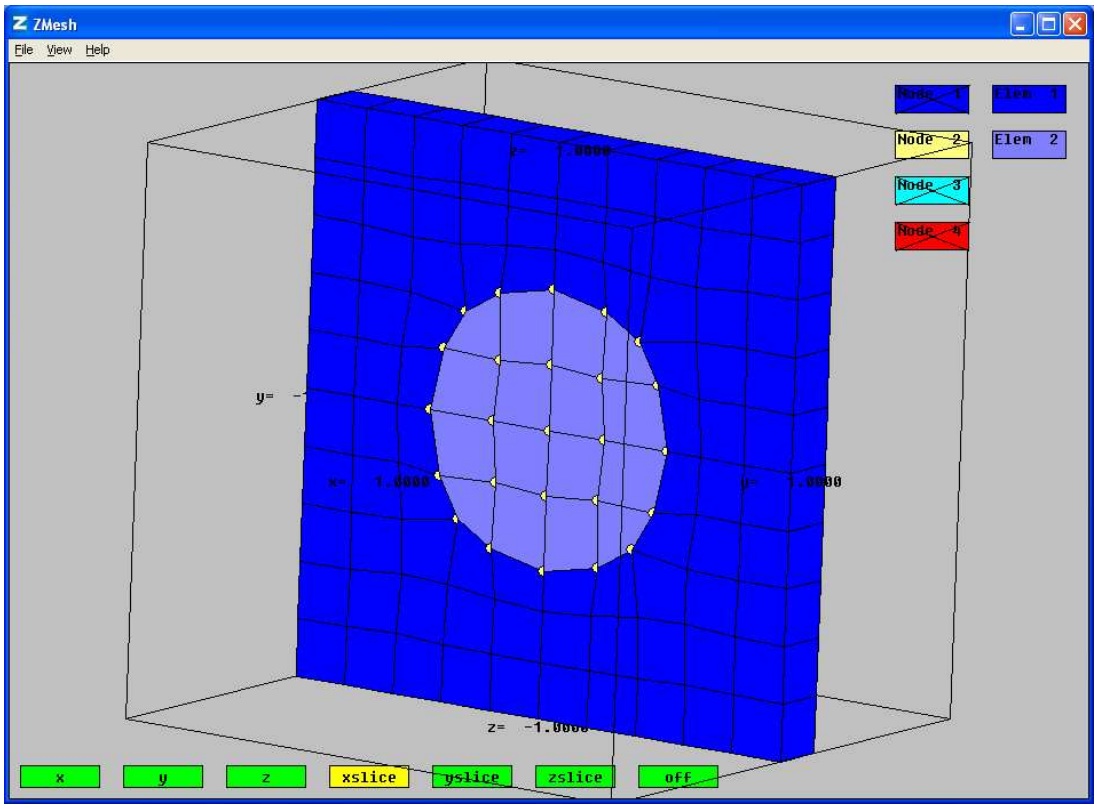


Fig.11: Example of a slice through Regions 1 and 2.

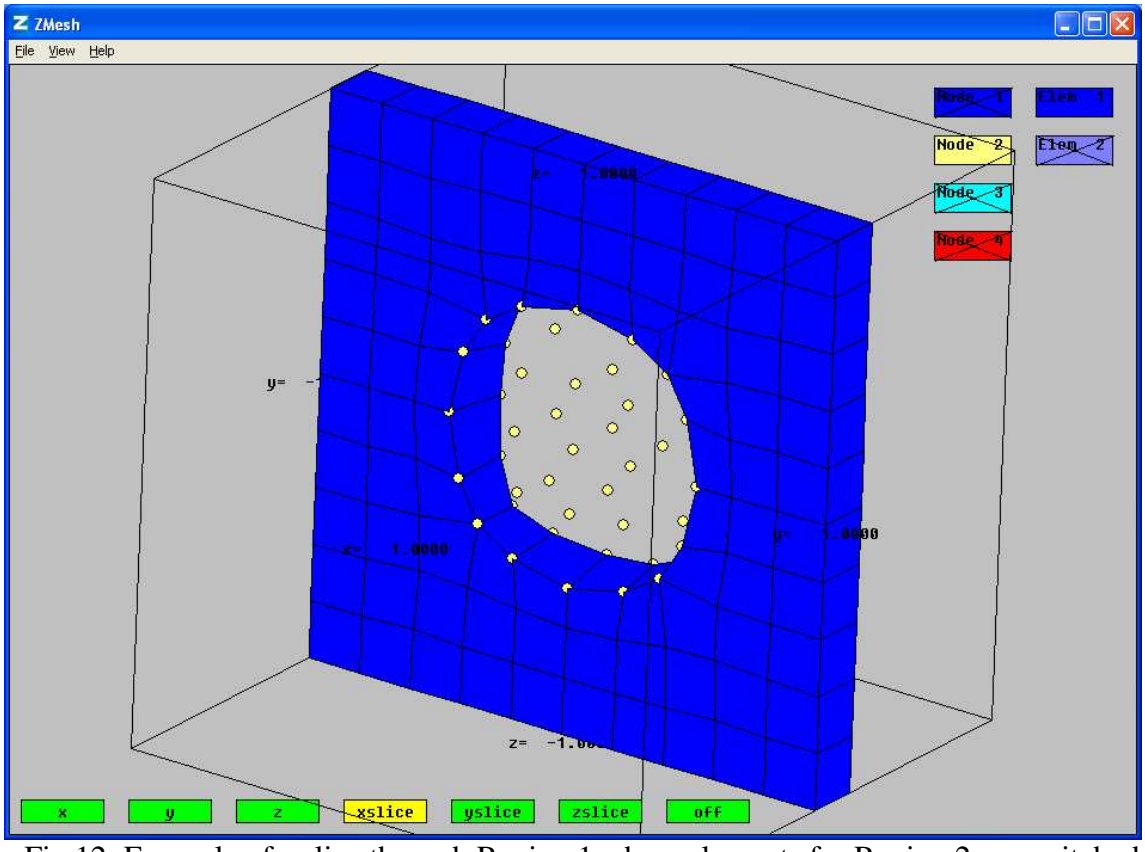


Fig.12: Example of a slice through Region 1 where elements for Region 2 are switched off.

In this case, the mtf file that is generated by Zmesh.exe is:

```
IMax:       10
JMax:       10
KMax:       10

    I       J       K         x           y           z      RegNo RegUp
=============================================================================
       0       0       0 -1.0000E+00 -1.0000E+00 -1.0000E+00     4     1
       1       0       0 -7.9998E-01 -1.0000E+00 -1.0000E+00     4     1
       2       0       0 -5.9995E-01 -1.0000E+00 -1.0000E+00     4     1
       3       0       0 -3.9991E-01 -1.0000E+00 -1.0000E+00     4     1
       4       0       0 -1.9992E-01 -1.0000E+00 -1.0000E+00     4     1
       5       0       0  2.2495E-05 -1.0000E+00 -1.0000E+00     4     1
       6       0       0  1.9996E-01 -1.0000E+00 -1.0000E+00     4     1
       7       0       0  3.9992E-01 -1.0000E+00 -1.0000E+00     4     1
       8       0       0  5.9993E-01 -1.0000E+00 -1.0000E+00     4     1
       9       0       0  7.9996E-01 -1.0000E+00 -1.0000E+00     4     1
      10       0       0  1.0000E+00 -1.0000E+00 -1.0000E+00     4     0
       0       1       0 -1.0000E+00 -7.9998E-01 -1.0000E+00     4     1

      Etc. ……
```

The MTF file is not mysterious, it is just a text file showing the locations of nodes, their Region, and their connectivity to neighbouring regions. This file is read in by the program Zincwin (see later). Normally, the user does not have to look at this file.

*Input file *.zin*

ZINC requires information that controls the performance of the software, materials data for each region, and data relating to boundary conditions that are not set by default. For the dielectric problem this information is given in the following *.zin file:

```
key_sim=1                  ! = 1 for static or steady/cyclic states, = 2 for transient problems
nvar=1                     ! No. of unknown variables (in this case just one: electrostatic potential)
nstep=10000                ! No. of iterations
omega=0.1                  ! Successive over-relaxation parameter (<1)  (Default 0.1)
nreg=4                     ! No. of Regions in model
key_rdu=0                  ! = 0 start from initial condition in *.zin (=1 continue from previous sim.)
scale=1                    ! scaling factor for lengths found in *.mtf
labels=V                   ! List of unknown variables (nvar in number)

region 1 elements 3 values C   !For elements of Region 1 the are 3 non-zero
                               !coefficients of ZINC matrix C
1 1 1 1 =   eps1
1 2 1 2 =   eps1
1 3 1 3 =   eps1

region 2 elements 3 values C   !For elements of Region 2 the are 3 non-zero
                               !coefficients of ZINC matrix C
1 1 1 1 =  eps2
1 2 1 2 =  eps2
1 3 1 3 =  eps2

region 3 nodes 1 values        !For nodes of Region 3 there is 1 value 1.0 for unknown variable
1 = 1.0
```

```
region 4 nodes 1 values          !For nodes of Region 4 there is 1 value 0.0 for unknown variable
1 = 0.0

Init                             !Initial value 0.0 for unknown variable
1 = 0.0
```

In this problem, as specified in Eq (8), the C matrix coefficients are just the permittivities. In this simple case, all regions are isotropic so permittivities 11=22=33. Note that coefficient 1111 is permittivity 11; 1212 is permittivity 22 etc. The first and third index numbers refer to the variable while the second and forth relate to spatial dimension (See User Manual, Chapter 4). In this case, there is only one variable so all indexes are 1x1y. It would, of course, be easy to simulate an anisotropic dielectric rather than an isotropic one as in this example.

The constants “eps1” and “eps2” are in the constants file, dielectric.con. They specify the permittivity of both regions. You can easily change these permittivities without altering the dielectric.zin file.

Running the simulation

Having prepared the Zinc input file, it is time to run the actual simulation using Zinc shown in Fig 13. Zinc also pops up an additional console window. As Zinc runs, information about convergence is written into the console window.

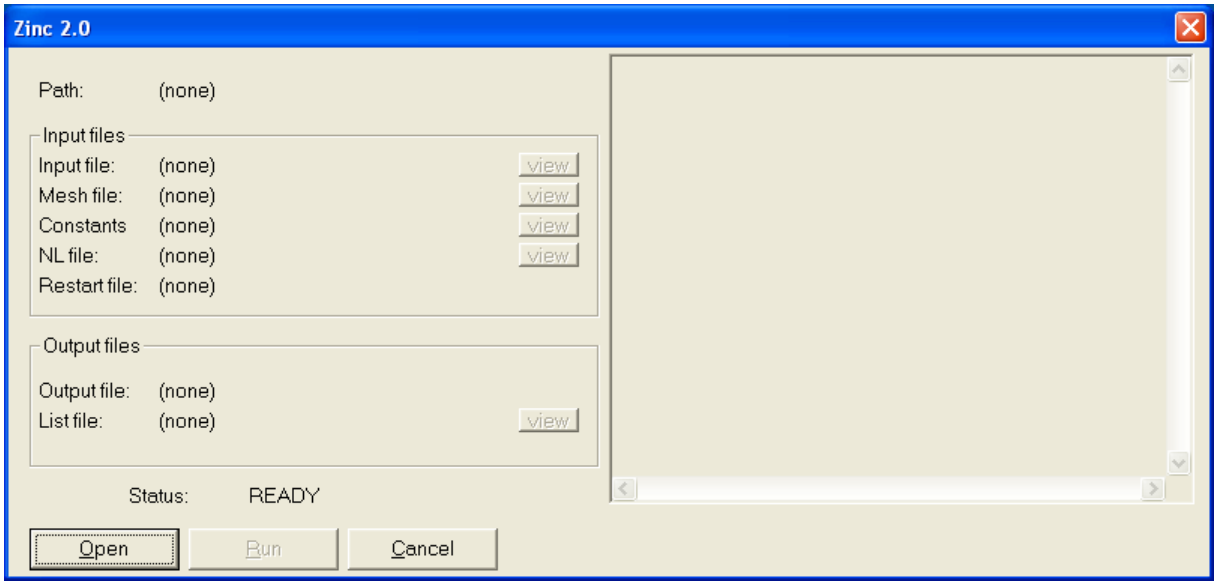


Fig 13: Zinc (core) window

Now, use the Open button to select the file dielectric.zin, Fig 14

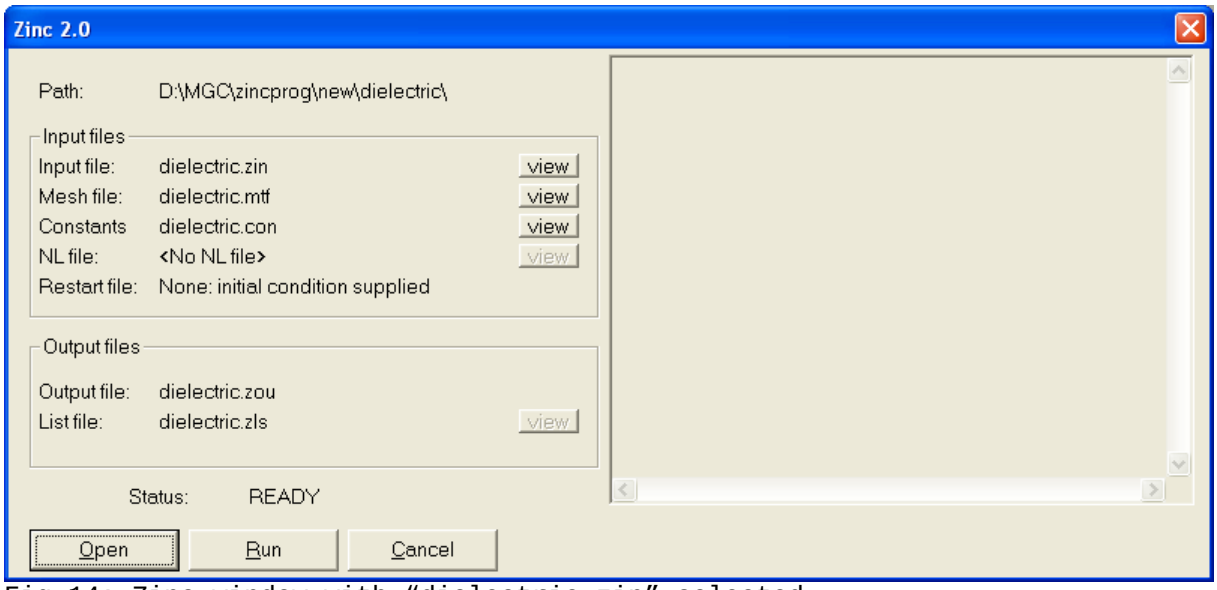


Fig 14: Zinc window with “dielectric.zin” selected.

The program checks all input files are present and shows the filename of each. To run Zinc, at least an input file (*.zin) and a mesh file (*.mtf) are needed. The constants and non-linearity (NL) files are optional. A restart file is needed only in the case key_rdu=1 (which means, “continue where you left off”. Since we’ve set this to zero (see above), this is not needed.

You can view any file but clicking on the “view” button next to it. In Fig 15, we are viewing the *.zin file. Note that the program does not allow you to edit files, you should use a text editor to do that. You can just use Notepad, but a programming text editor is probably better. We personally use Emacs, but there are many free editors to chose from.

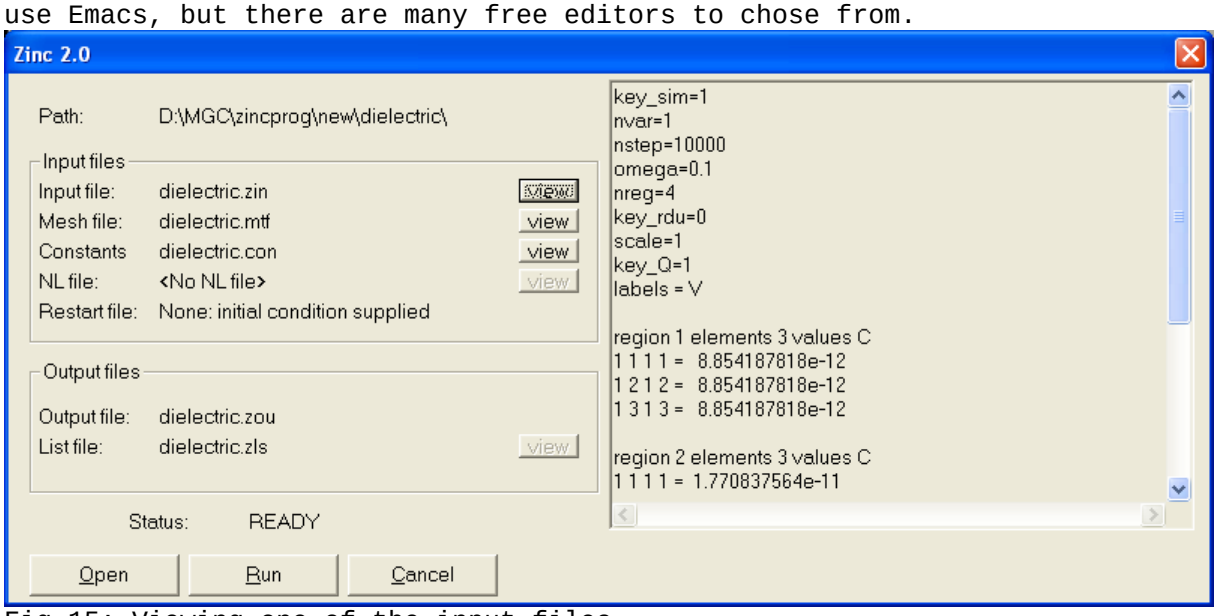


Fig 15: Viewing one of the input files

If all seems to be in order it is now time to run Zinc. Press the “run” button. In the console window you will see output shown in Fig 16.

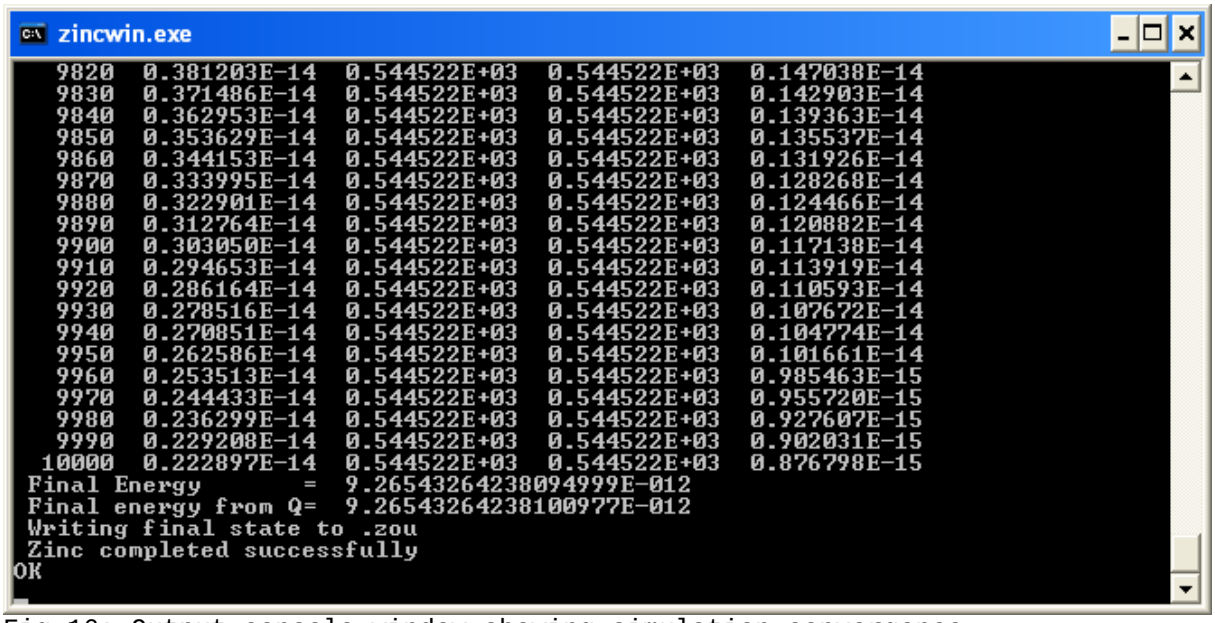


Fig 16: Output console window showing simulation convergence.

The number of time steps is shown in the first column. Here we are running up to 10,000 time steps. The second and fifth columns show numbers which should get low when an approximate solution has been found to the problem. The 3rd and 4th columns show a number which approximates the "left hand side" and "right hand side" of the set of equations to be solved. These numbers should be equal to each other. Clearly in this simulation, a solution has been found to good accuracy. The final energy of the system is output using two calculations. These numbers should be very similar to each other. If they are not an error has occurred.

Viewing the output.

Once the simulation has completed, the Status shown returns to "READY" and the Zinc list file *.zls becomes viewable (Fig 17). This file contains information about Zinc's input reflecting the specifications in the *.zin file. Note that all the components of the Zinc matrices **c, a, f, g, q** are listed in this file, including the zero ones so the user can check what was actually set up (recall that we only set the **c** values in the *.zin file so all other matrices should be full of zeros in this simple system)

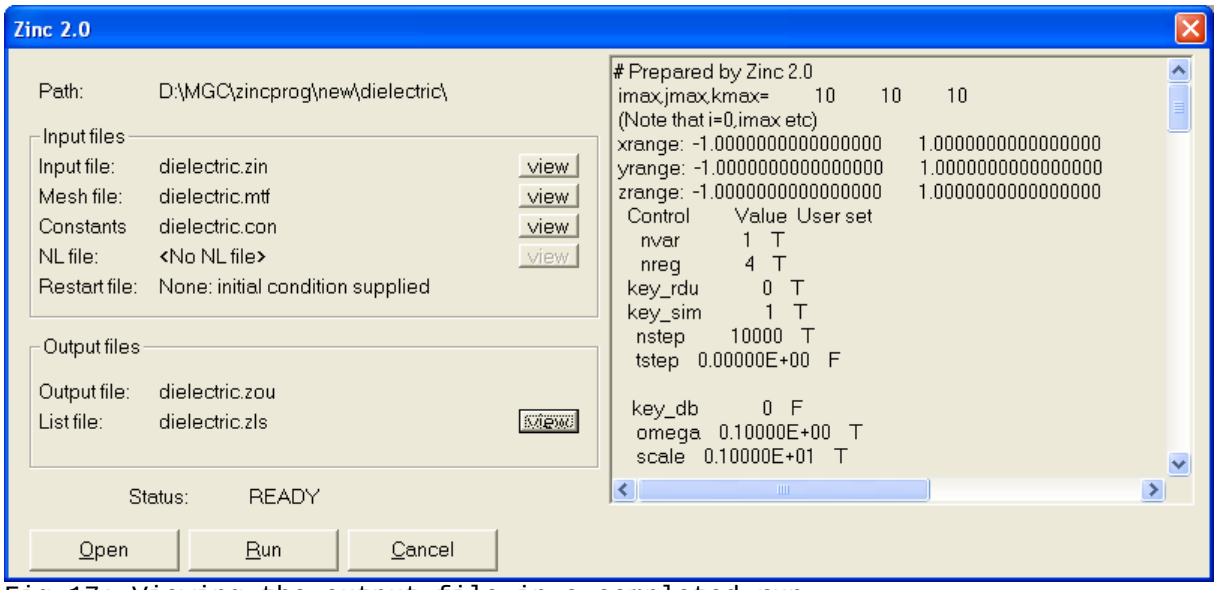


Fig 17: Viewing the output file in a completed run

As Zinc shows us, the output file containing the simulation result in *.zou. This file is not viewable, because it is not intended to be human readable. There's no mystery involved, however, and you can look at this file using a text editor.

Note that the output files are automatically saved to disk with the names *.zou, *.zls. There is no need to save the files.

Zinc is now complete and you dismiss the window by pressing Cancel.

A quick look at the result

We can get a simple view of the data using Zmesh. In Zmesh, open the zinc output file, dielectric.zou using the File > Open ZOU option as shown in Fig 17. We can see from the figure that the boundary conditions V=1 and V=0 Volts have been implemented. In systems with more than one variable, you can switch between the different variables using buttons at the top-left of the screen. The usual features to hide regions and take slices are still available. Fig 18 shows a slice through the data. Note that the inner region (region 2) has been set to show the region while the outer region is showing the data. Thus, the colour key elements on the right of the screen now have three states on / DATA / off.

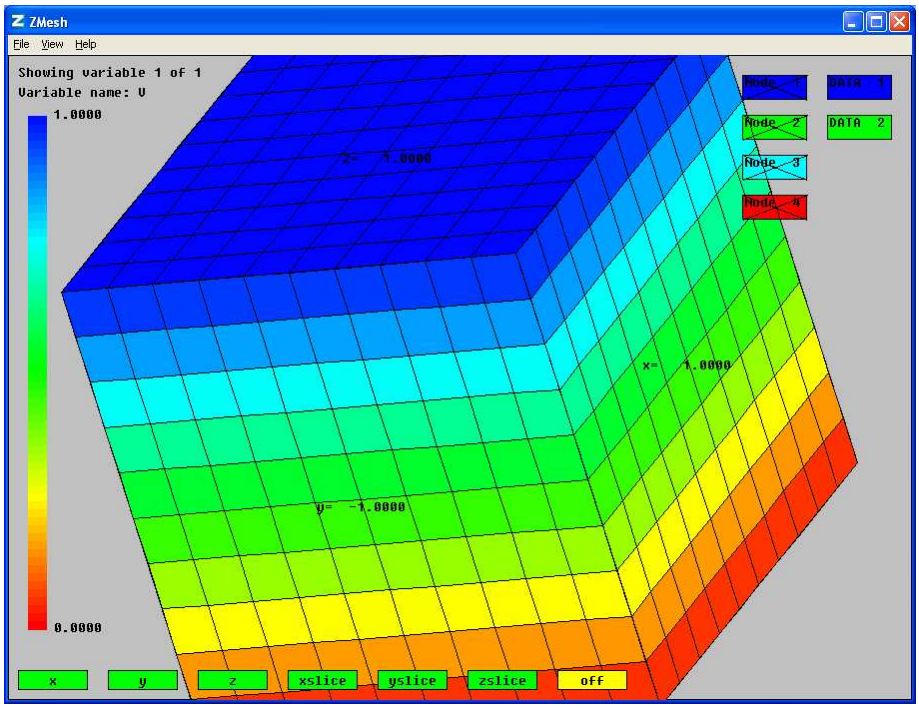


Fig 17: Viewing the simulation output (ZOU) file

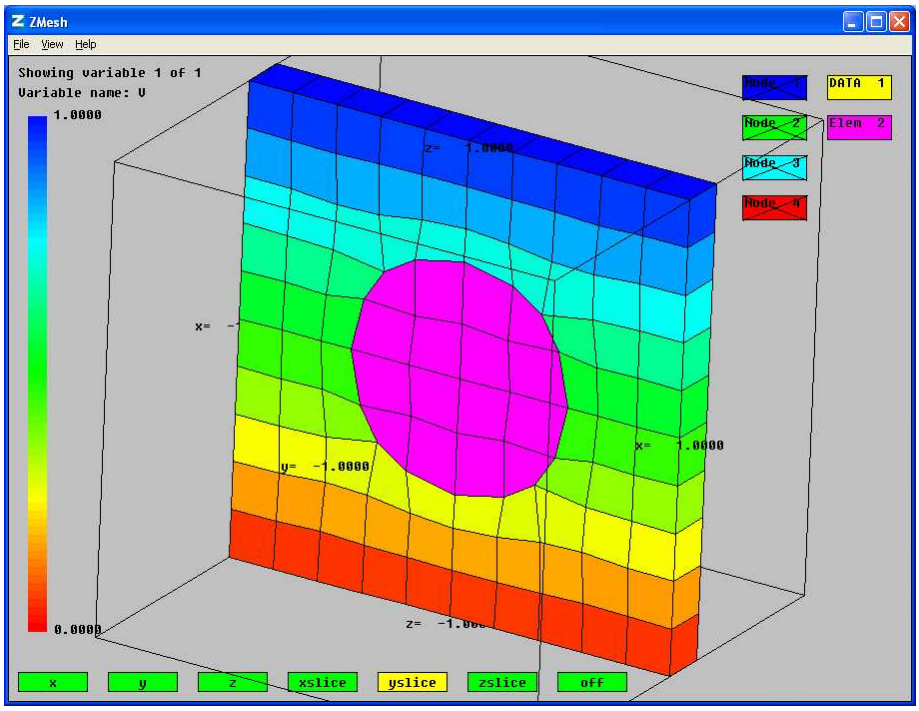


Fig 18: Slice view through the data showing simulation result in region 1.

Post processing

We now wish to examine the output of the simulation using ZPP (the Zinc post processor). This program allows us to take various “scans” through the data and prepare graphs either of the simulated quantities (electric potential in this case) or quantities derived from these. In a particular, we will show how to calculate the E-field and D-field.

Open a new text file called *.zpp (Zinc post processor) and type in the following.

```
key_plot=2               ! output emf files
ftol=1e-3                ! tolerance for fitting

!          N       x1   y1    z1       x2    y2    z2

linescan 100    0.101 -0.3  -1.0     0.101 -0.3  +1.0 = V
linescan 100    0.101 -0.3  -1.0     0.101 -0.3  +1.0 = V^2
linescan 100    0.101 -0.3  -1.0     0.101 -0.3  +1.0 = -Vz
linescan 100    0.0    0.0  -1.0     0.0    0.0  +1.0 = -Vz
linescan 100    0.0    0.0  -1.0     0.0    0.0  +1.0 = &
-{1=eps1,2=eps2}*Vz

!         yplane    y     x1     x2      z1     z2    Nx  Nz

planescan      2  0.0   -1.0    1.0    -1.0    1.0    40  40 = V
planescan      2  0.0   -1.0    1.0    -1.0    1.0    40  40 = &
-{1=eps1,2=eps2}*Vz
```

Here we are requesting 7 scans, 5 linescans and 2 planescans. Linescans are cuts through the data along a line. The first one is a line scan with 100 points between coordinates (0.101,-0.3,-1.0) and (0.101,-0.3,+1.0), i.e. it is a scan along the z-direction. The function to be plotted is shown after the equals sign, V, in the case, the electrostatic potential. Zppwin knows about "V" because it also reads the *.zin input file in which, we recall, we put the command

```
Labels = V
```

The labels command is optional: if you did not use it you'd have to refer to u1 instead of V. In multi-variable simulations, the default names are u1, u2 etc. Clearly it is more readable to specify a mnemonic name using "labels".

The file illustrates the use of comments (with !) and the continuation character (&) for splitting lines which are uncomfortably long. Whether split or not, each scan command should not be more than 1000 characters long.

Having prepared the *.zpp file, launch ZPP and select the file using the "Open" button as shown in Fig 19

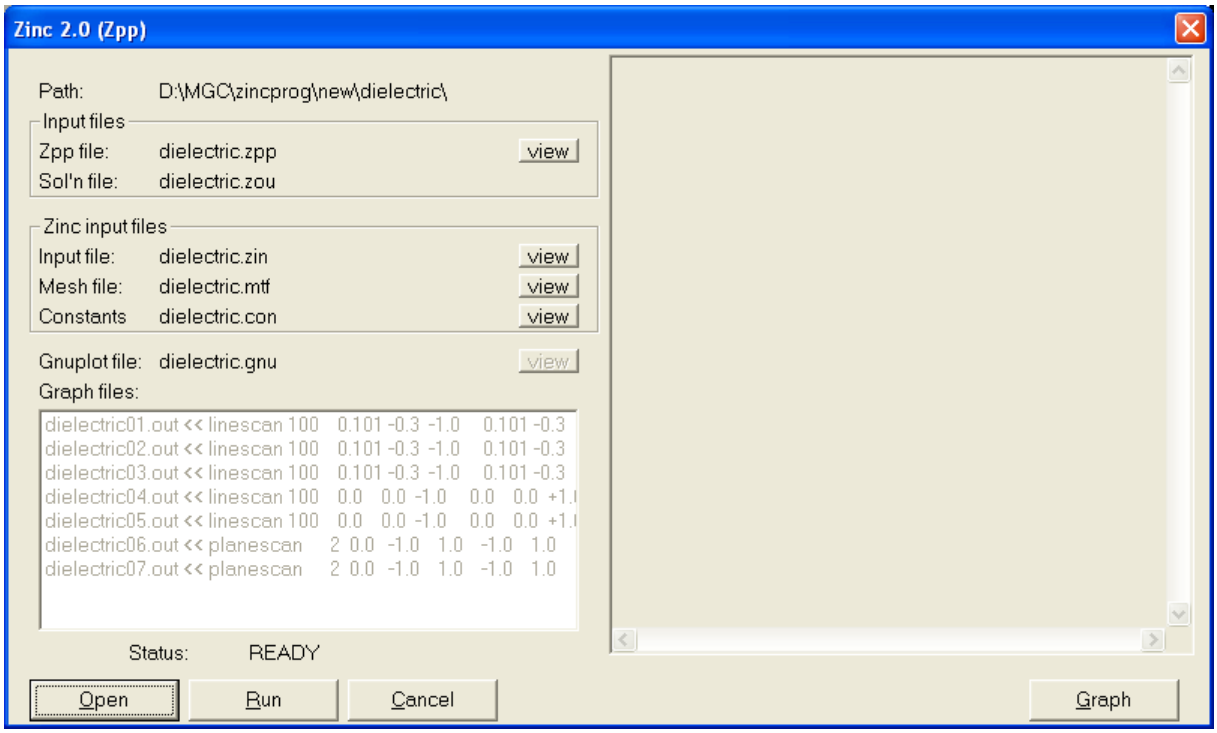
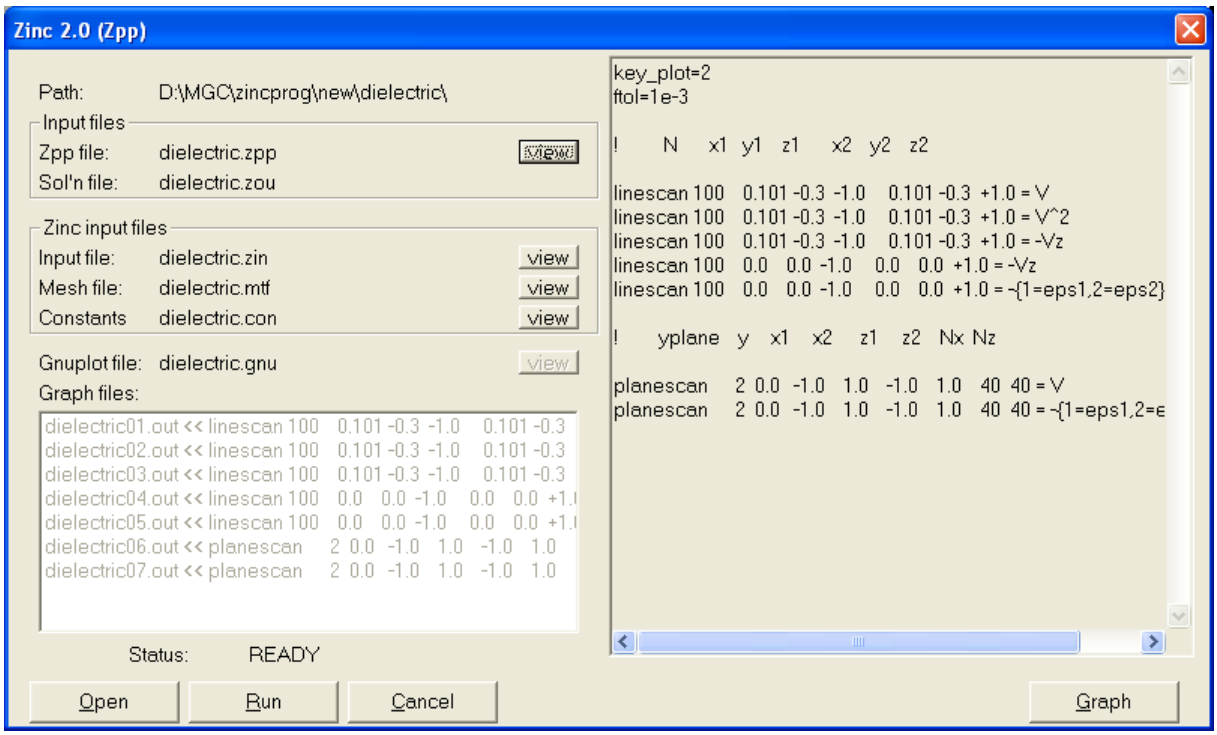


Fig 19: Creating scans through the data using ZPP

The main input files used by zppwin are *.zpp and *.zou (the simulation solution). Zpp will complain if any of these are absent. Zpp also reads *.zin, *.mtf *.con as used by Zinc so these must also be present. As before we can examine files, using the view button as shown in Fig 20


Zinc 2.0 (Zpp)
Path: D:\MGC\zincprog\new\dielectric\
Input files
Zpp file: dielectric.zpp
view
Sol'n file: dielectric.zou
Zinc input files
Input file: dielectric.zin
view
Mesh file: dielectric.mtf
view
Constants dielectric.con
view
Gnuplot file: dielectric.gnu
view
Graph files:
dielectric01.out << linescan 100 0.101 -0.3 -1.0 0.101 -0.3
dielectric02.out << linescan 100 0.101 -0.3 -1.0 0.101 -0.3
dielectric03.out << linescan 100 0.101 -0.3 -1.0 0.101 -0.3
dielectric04.out << linescan 100 0.0 0.0 -1.0 0.0 0.0
dielectric05.out << linescan 100 0.0 0.0 -1.0 0.0 0.0
dielectric06.out << planescan 2 0.0 -1.0 1.0 -1.0 1.0
dielectric07.out << planescan 2 0.0 -1.0 1.0 -1.0 1.0
Status: READY
Open
Run
Cancel
Graph
key_plot=2
ftol=1e-3
! N x1 y1 z1 x2 y2 z2
linescan 100 0.101 -0.3 -1.0 0.101 -0.3 +1.0 = V
linescan 100 0.101 -0.3 -1.0 0.101 -0.3 +1.0 = V^2
linescan 100 0.101 -0.3 -1.0 0.101 -0.3 +1.0 = -Vz
linescan 100 0.0 0.0 -1.0 0.0 0.0 +1.0 = -Vz
linescan 100 0.0 0.0 -1.0 0.0 0.0 +1.0 = -{1=eps1,2=eps2}
! yplane y x1 x2 z1 z2 Nx Nz
planescan 2 0.0 -1.0 1.0 -1.0 1.0 40 40 = V
planescan 2 0.0 -1.0 1.0 -1.0 1.0 40 40 = -{1=eps1,2=e
Fig 20: Viewing the dielectric.zpp input file which specifies line and plane scans.

If all is well, run the post processor using “Run”. Once the post processing is complete, the “graph files” list becomes active. This list shows us which datafiles correspond to which of

the selected scans. For example dielectric01.out will contain the first linescan requested. Clicking on this line, as illustrated in Fig 21, allows us to view the file.

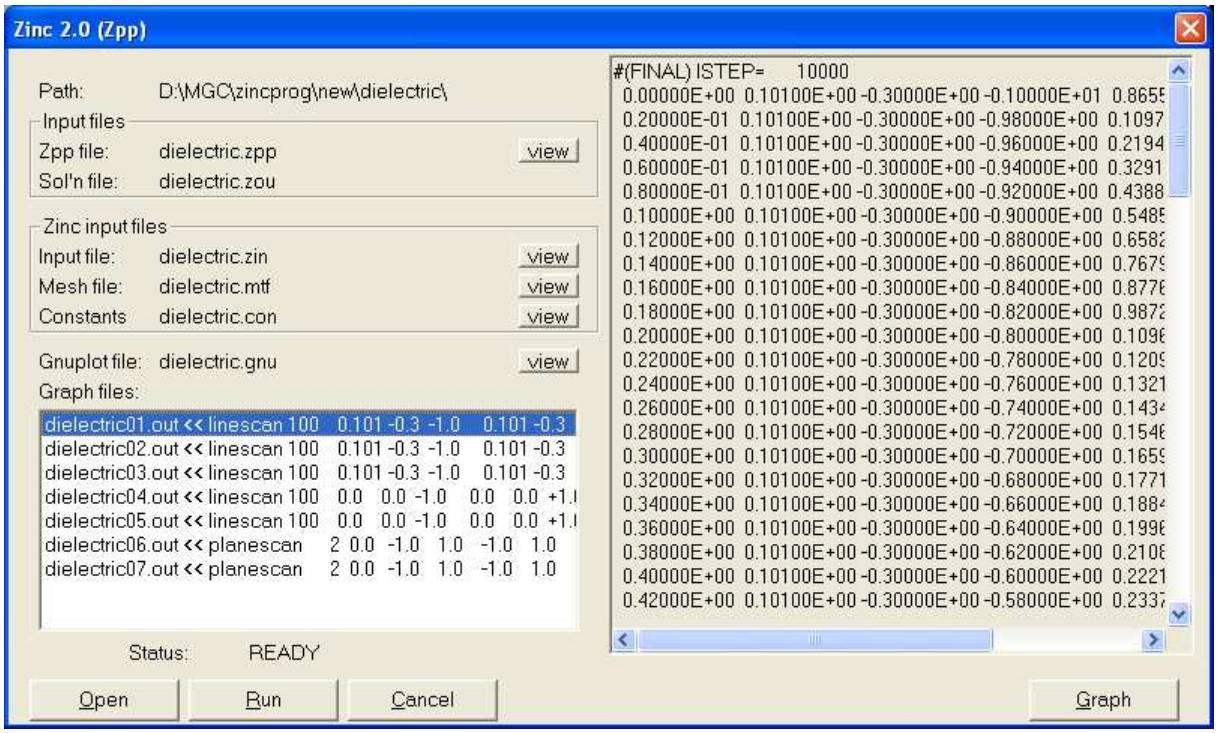


Fig 21: Once a run is complete the “Graph files” list is active and we see the resulting graphs

But it’s much more fun to view this as a graph. To do this, click “Graph” at the bottom right of the window. Another window pops up, Fig 22, showing the graph. In this case, it is a graph of voltage versus distance along the z-axis. The voltage goes from 0 to 1 V as we expect and there is a “kink” in the middle where the high permittivity sphere is located.

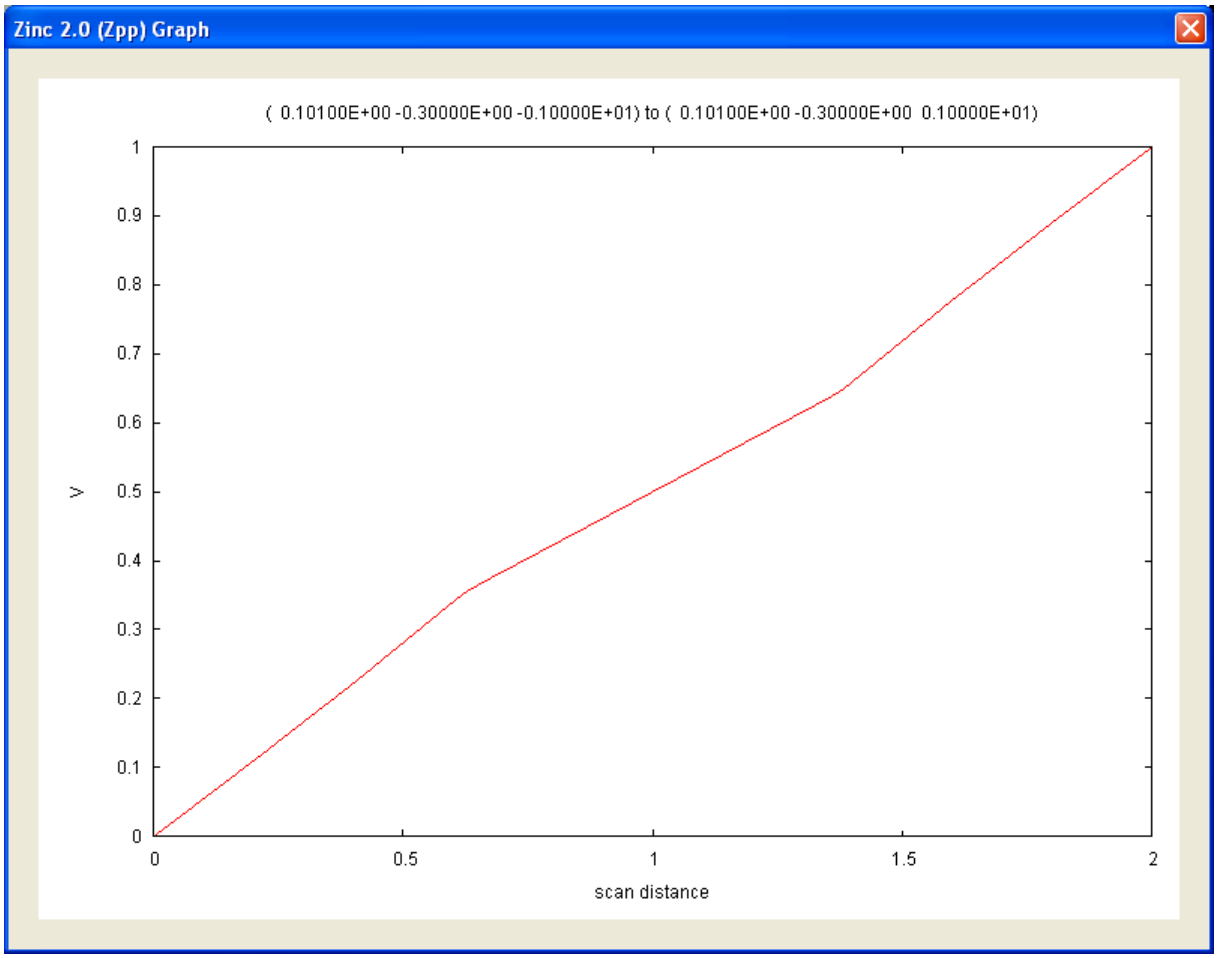


Fig 22: Variation of voltage through the sample

You can look at the other graphs using the Next or Prev buttons on the graph window, or by returning to the listbox in the zppwin window. Note that there is no need to save any of these files: they are automatically saved when we run zppwin. In fact two types of files are saved in this case, the dielecticxx.out files which contain a table of numbers corresponding to the scan and the dielectricxx.emf which contain the actual graphs. Graph files are only generated if key_plot is 1 or 2. key_plot=1 generates encapsulated postscript files (eps) while key_plot=2 (as used here) generates enhanced metafiles (emf). Graph previews within zppwin are only available for emf files, key_plot=2. To view eps files, you'll need to install ghostview, a free program. It depends which word processor, you intend to import graphs to. Eps graphs are suitable for LaTeX, while emf are suitable for Microsoft products, like Word and Powerpoint. Both these graph formats are vector based.

Fig 22 is the second of the two plane scans showing D_z, the z component of the D-field.

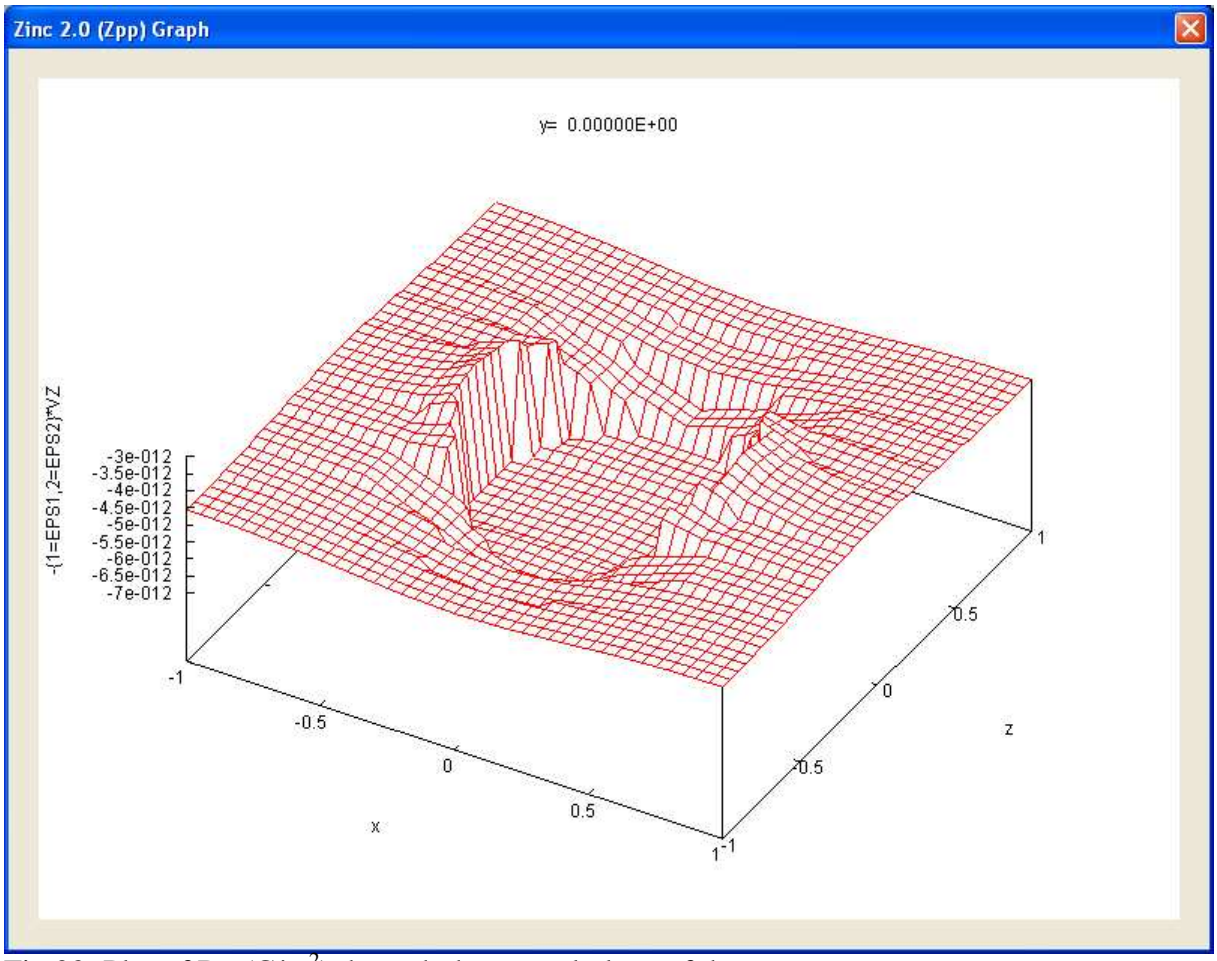


Fig 23: Plot of Dz (C/m^2) through the central plane of the system

Let us look at the command which generated this plot:

```
planescan      2  0.0   -1.0    1.0    -1.0    1.0    40  40 = &
-{1=eps1,2=eps2}*Vz
```

The {1=eps1,2=eps2} expressions means
"if we are in region 1, use value eps1; if we are in region 2, use value eps2".

Here eps1 and eps2 are parameters appearing in the *.con file:

```
eps0=8.854e-12
eps1=eps0
eps2=eps0*2
```

As we scan in the plane indicated we will encounter region 1 and region 2 (inner sphere) areas. The appropriate value of permittivity needs to be taken in each case. The expression is then -eps*Vz. Vz means dV/dz. This is the standard notation used by Zinc: since we have given potential the name V, the derivatives can be accessed using "Vx, Vy, Vz". Thus, the whole expression is eps*Ez=Dz, the z-component of displacement field.

The parameters after the planescan command indicate what sort of scan is needed. "2" means perpendicular to the y-direction. –1.0 1.0 means scan x between [-1.0,1.0]. The next "-1.0 1.0" means z=[-1.0,1.0]. "40 40" is the number of points requested along each direction.

The meaning of the other scans should now be apparent. Note that quite arbitrary functions involving the variables solved for and the constants in `*.con` can be used. The second line scan plots V^2 which is the potential squared. All the usual operators are allowed and the functions sin, cos, tan, exp, log etc can be used also. (see the Zinc User Manual, Chapter 5 for more information)

Traceability

Before continuing this tutorial, a quick word about traceability. In running the dielectric simulation we have generated 33 files:

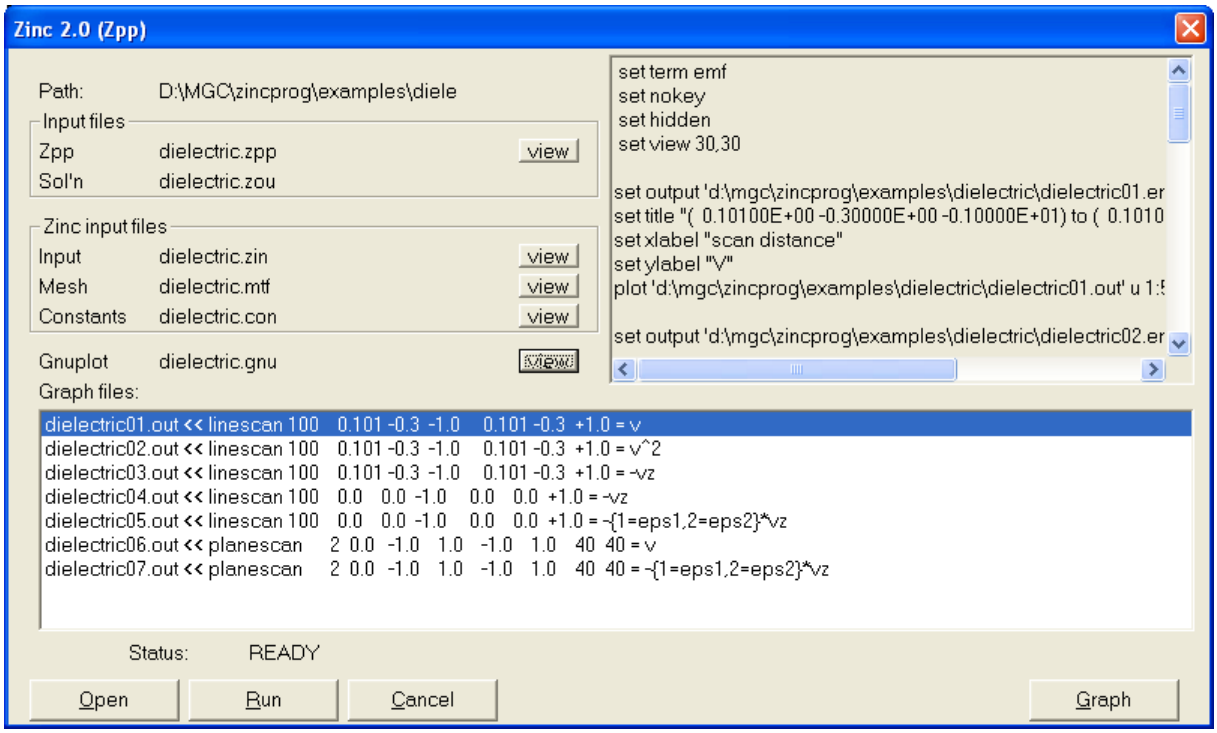
```
Dielectric.zin
Dielectric.min
Dielectric.mtf
Dielectric.con
Dielectric.mls
Dielectric.zls
Dielectric.zou
Dielectric.zpp
Dielectric.gnu
Dielectricxx.out   xx=01 to 10
Dielectricxx.emf   xx=01 to 10
```

This might seem like rather a lot of files but this structure provides traceability. Imagine 6 months down the line you look at file dielectric01.emf and wonder "how on earth did I generate this graph?". By looking at dielectric.zpp, we can see that it is a line scan between such-and-such points of the electric potential, V. But how did we get this simulation in the first place? By looking at dielectric.zin (physics) and dielectric.min (geometry) we can see exactly how the simulation was run. We can then reproduce the same simulation or a variation of it easily. Thus, the Zinc, text-based, file based paradigm give clear traceability and is superior to the "nested dialog box" approach used by most FE packages. With these other packages, it is common to end up with graph picture files and not know how you generated them.

More advanced graph plotting

We've seen that Zpp can generate basic graphs which give a good immediate insight into what is happening in the simulation. For publication, however, these graphs are probably not sufficient. For a start it will be necessary to re-label the y-axis of most graphs. Labels like `-{1=eps1,2=eps2}*Vz` don't mean much to most people! Zinc, of course, just labels the axes with the formula you give it: it doesn't know the physics, nor the right units to use. Beyond this it might be necessary to compare to curves on the same graph with a legend to distinguish them. Or, we might want to compare FE simulation results with theory or experimental data obtained from elsewhere.

Zinc creates its graphs by running gnuplot, an open source graph plotting package bundled with the Zinc distribution. The script file used to generate the graphs is shown called *.gnu and can be viewed by pressing the view button, shown in Fig 24.


Zinc 2.0 (Zpp)
Path: D:\MGC\zincprog\examples\diele
Input files
Zpp dielectric.zpp view
Sol'n dielectric.zou
Zinc input files
Input dielectric.zin view
Mesh dielectric.mtf view
Constants dielectric.con view
Gnuplot dielectric.gnu view
Graph files:
dielectric01.out << linescan 100 0.101 -0.3 -1.0 0.101 -0.3 +1.0 = v
dielectric02.out << linescan 100 0.101 -0.3 -1.0 0.101 -0.3 +1.0 = v^2
dielectric03.out << linescan 100 0.101 -0.3 -1.0 0.101 -0.3 +1.0 = -vz
dielectric04.out << linescan 100 0.0 0.0 -1.0 0.0 0.0 +1.0 = -vz
dielectric05.out << linescan 100 0.0 0.0 -1.0 0.0 0.0 +1.0 = -{1=eps1,2=eps2}*vz
dielectric06.out << planescan 2 0.0 -1.0 1.0 -1.0 1.0 40 40 = v
dielectric07.out << planescan 2 0.0 -1.0 1.0 -1.0 1.0 40 40 = -{1=eps1,2=eps2}*vz
set term emf
set nokey
set hidden
set view 30,30
set output 'd:\mgc\zincprog\examples\dielectric\dielectric01.er
set title "(0.10100E+00 -0.30000E+00 -0.10000E+01) to (0.1010
set xlabel "scan distance"
set ylabel "V"
plot 'd:\mgc\zincprog\examples\dielectric\dielectric01.out' u 1:
set output 'd:\mgc\zincprog\examples\dielectric\dielectric02.er
Status: READY
Open
Run
Cancel
Graph
Fig 24: The gnuplot script file used to generate the graphs

The gnuplot file consists of a series of commands most of which revolve around the “plot” command. There are also lots of “set” commands. In this case we are setting the “term(inal)” to be emf so that gnuplot will output enhanced metafiles.

Let’s consider the first plot command in this file

```
set output 'd:\mgc\zincprog\examples\dielectric\dielectric01.emf'
set title "(  0.10100E+00 -0.30000E+00 -0.10000E+01) to (  0.10100E+00 -
0.30000E+00  0.10000E+01)"
set xlabel "scan distance"
set ylabel "V"
plot 'd:\mgc\zincprog\examples\dielectric\dielectric01.out' u 1:5 w l
```

This instructs gnuplot to plot the file dielectric01.out to make graph file dielectric01.emf. “u 1:5” means use columns 1 and 5 (resp.) for the x and y axes. “w l” means “with lines” ie datapoints are connected using lines. For more information about the gnuplot commands, see the gnuplot manual and tutorials. However, a lot can be gleaned from the default gnuplot script file generated by zppwin.

So how do we improve the graph quality? All gnuplot does it plot the *.out files. Since these files are still present on disk, we can replot these files in different ways as needed. The best way is to create a new gnuplot script file and run it through gnuplot. [In principle, one could just alter dielectric.gnu (in this example), but this file will be overwritten next time zppwin is run so its probably better to use a new file.] We will now prepare a file to

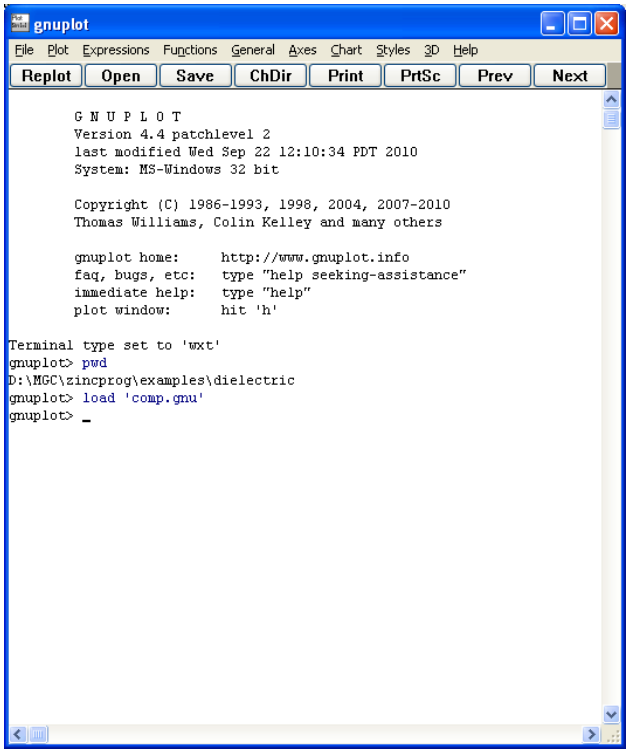
1) plot the first linescan with better labelled axes
2) plot the two Ez linescans in the same graph with a legend
3) plot the Dz planescan as a colour grid rather than a surface plot.

Create a text file called “comp.gnu” and enter the following

```
set term emf mono dashed

set output 'vscan.emf'
set xlabel 'Scan distance (m)'
set ylabel 'V (Volts)'
unset key

plot 'dielectric01.out' u 1:5 w l

set output 'Ezscan.emf'
set ylabel 'Ez (V/m)'
set key

plot 'dielectric03.out' u 1:5 t 'x=0.1 m' w l,\
'dielectric04.out' u 1:5 t 'x=0.0 m' w l

set output 'Dzmap.emf'
set title 'Dz (C/m^2)'
set xlabel 'x (m)'
set ylabel 'z (m)'
set size square
set pm3d map
splot 'dielectric07.out' u 1:3:4
```

By now, you should have a rough idea of what this is doing, but consult the gnuplot manual for further details. We are simply plotting the files dielectric01.out, dielectric03.out, dielectric04.out and dielectric07.out with slightly more advanced formatting.

There are several ways to run gnuplot. If you are unfamiliar with the command prompt, the simplest is to start gnuplot from its desktop/start menu shortcut producing the window in Fig 25.


gnuplot
File Plot Expressions Functions General Axes Chart Styles 3D Help
Replot Open Save ChDir Print PrtSc Prev Next
G N U P L O T
Version 4.4 patchlevel 2
last modified Wed Sep 22 12:10:34 PDT 2010
System: MS-Windows 32 bit
Copyright (C) 1986-1993, 1998, 2004, 2007-2010
Thomas Williams, Colin Kelley and many others
gnuplot home: http://www.gnuplot.info
faq, bugs, etc: type "help seeking-assistance"
immediate help: type "help"
plot window: hit 'h'
Terminal type set to 'wxt'
gnuplot> pwd
D:\MGC\zincprog\examples\dielectric
gnuplot> load 'comp.gnu'
gnuplot> _
Fig 25: The gnuplot window

First make sure you are in the right directory using the `pwd` command (as shown) and change directory using the `cd` command if necessary. Then use the load command to run the script file you've prepared (comp.gnu in this case). If all goes well the graphs are created in the same directory.

Tip: Note that we did not type in the full file path for the *.out files in comp.gnu. This is because gnuplot works in the current directory by default.

If you are familiar with using the command prompt, you can simply run gnuplot at the command line with the script file as an argument, fig 26

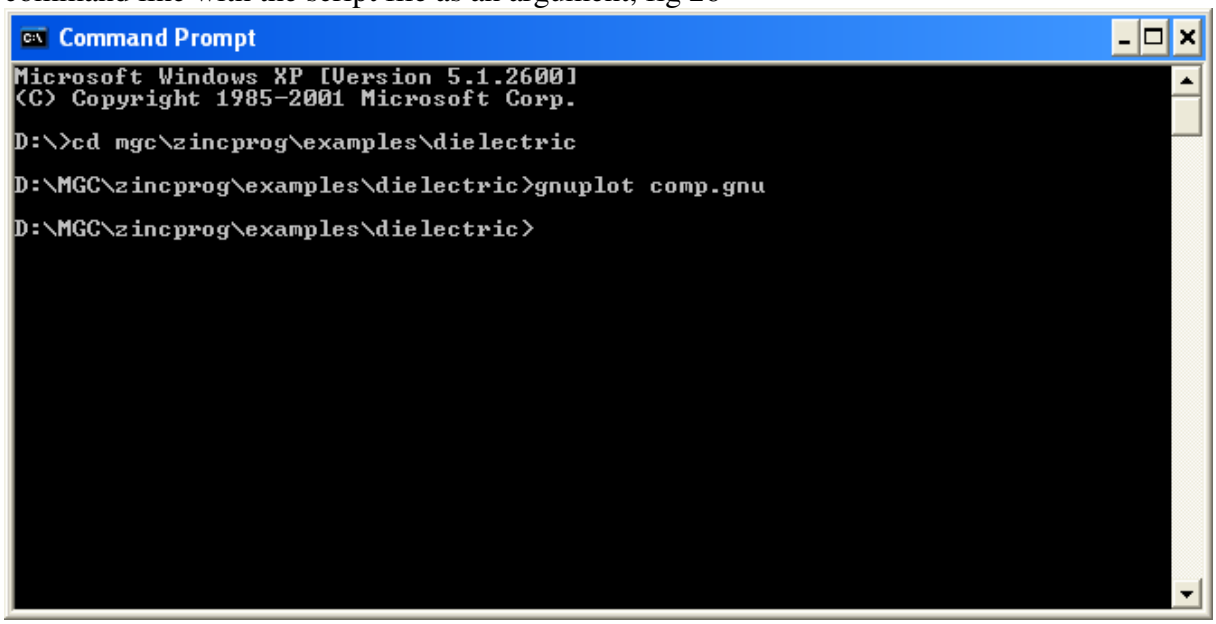


Fig 26: Running gnuplot under batch control

If all goes well the graph files are silently created (in the same directory). The three graphs generated are shown in Figs 27-29.

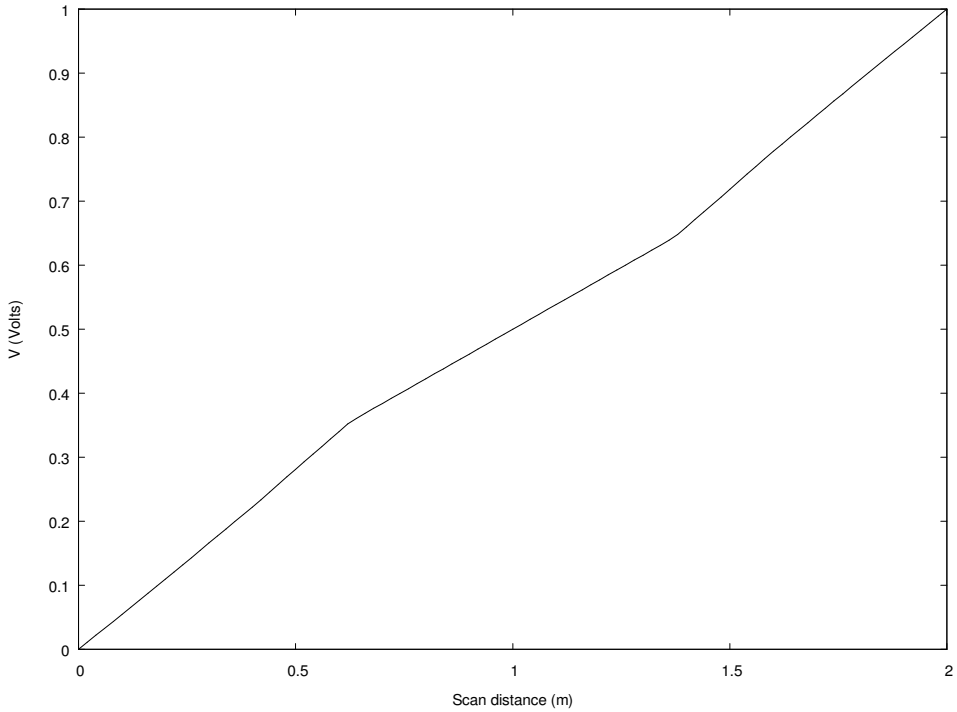


Fig 27: Graph plotting with better axis labelling

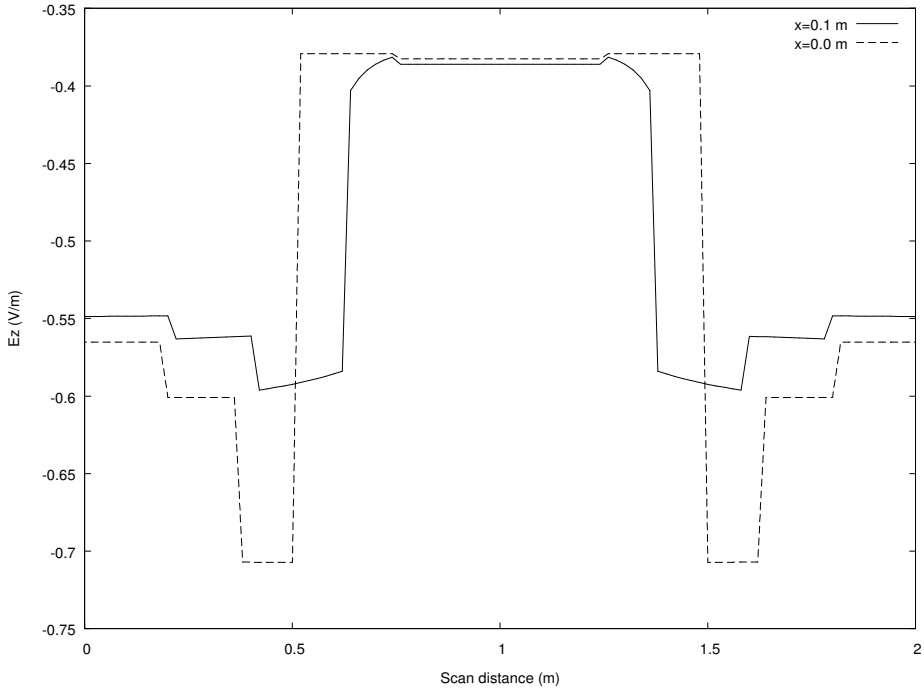


Fig 28: Graph with two curves and a legend.

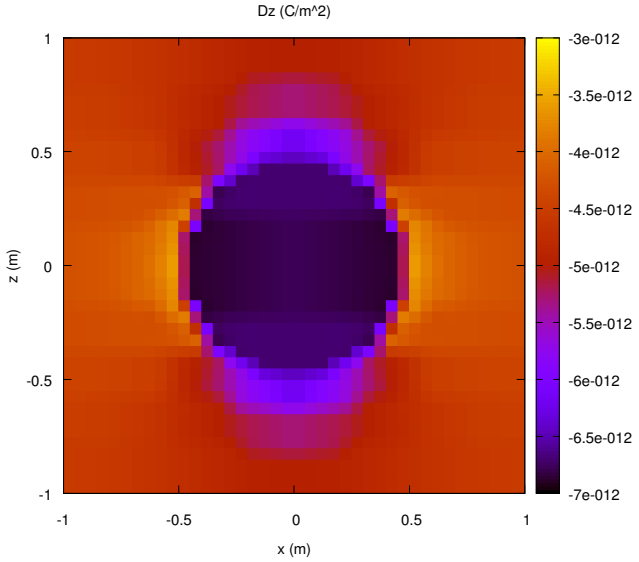


Fig 29: Colour coded graph of Dz

Figs 27 and 28 were prepared in black and white (with dashed lines) which is more usual for journal papers.

This is just a taste of the graphs one can prepare using gnuplot. The full range of graphs commonly found in papers (inset graphs, multiple axes, histograms, vector fields, error bars etc) can all be generated using gnuplot.

For completeness, Fig 30 shows the same 3 graphs rendered as eps (encapsulated postscript) and being viewed using ghostview (freeware). EPS is the normal graph format when using LaTeX: it arguably gives the best quality graphs.

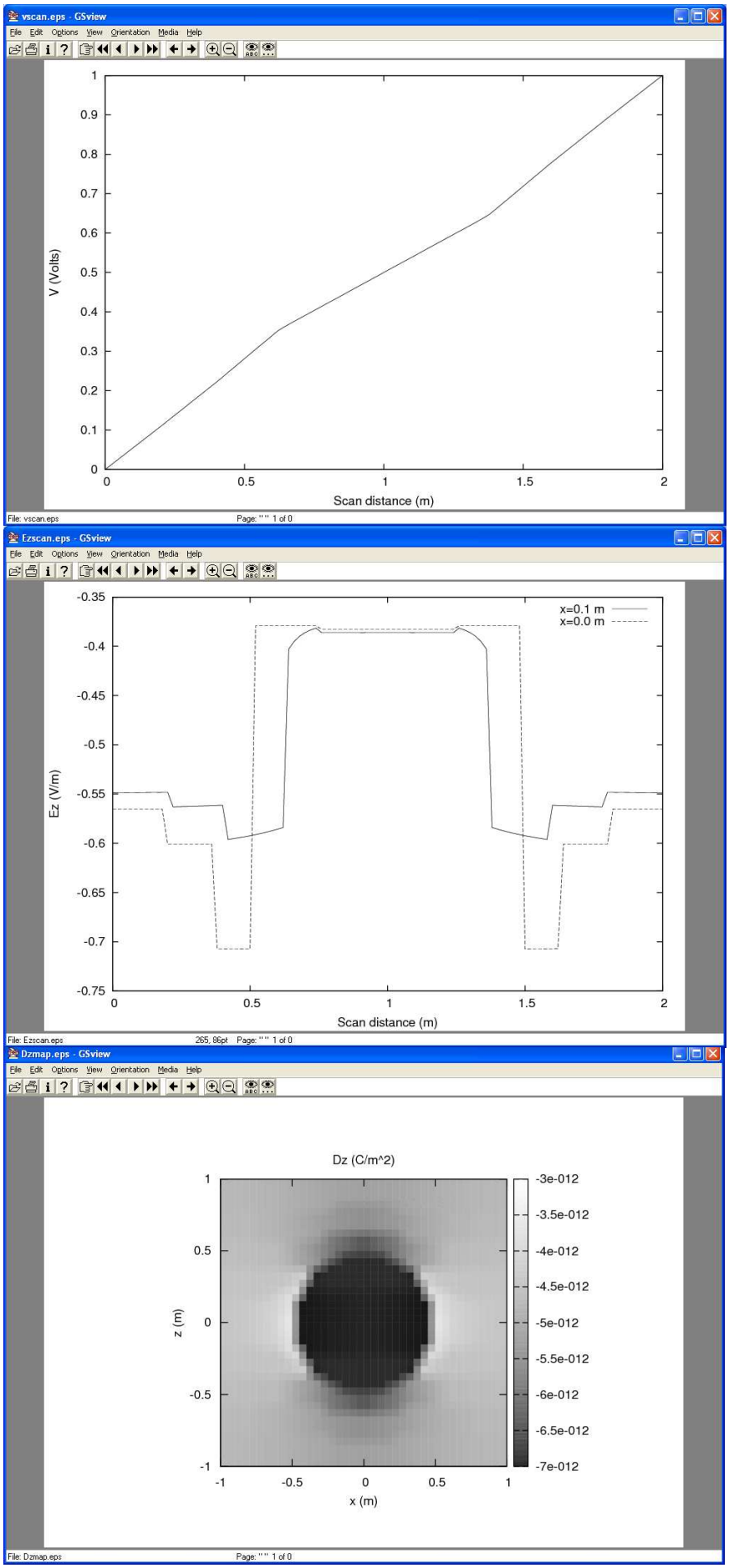


Fig 30: Encapsulated postscript graphs generated by Zinc using key_plot=1

If you don't like gnuplot

You don't have to use it! Since the data being plotted is created and stored as *.out files, you can plot it using any spreadsheet program or graph plotter. Or you could use scripting languages like Matlab to read the *.out files and prepare graphs. It's also possible to get Matlab to write Zinc's input files, call Zinc and read in and plot the resulting graphs.

Also in the directory

Dielslap.zin: Exactly the same simulation but with key_sim=3. That is, diagonally scaled GMRES solver.

Example 2: piezoelectric example

Directory: examples/piezo

Introduction

We now consider a piezoelectric problem. Here there are 4 variables to solve for, the x, y, z components of elastic displacement (ux, uy, uz) and the electrostatic potential (V). This is our first multiphysics problem. However, note that the problem is linear.

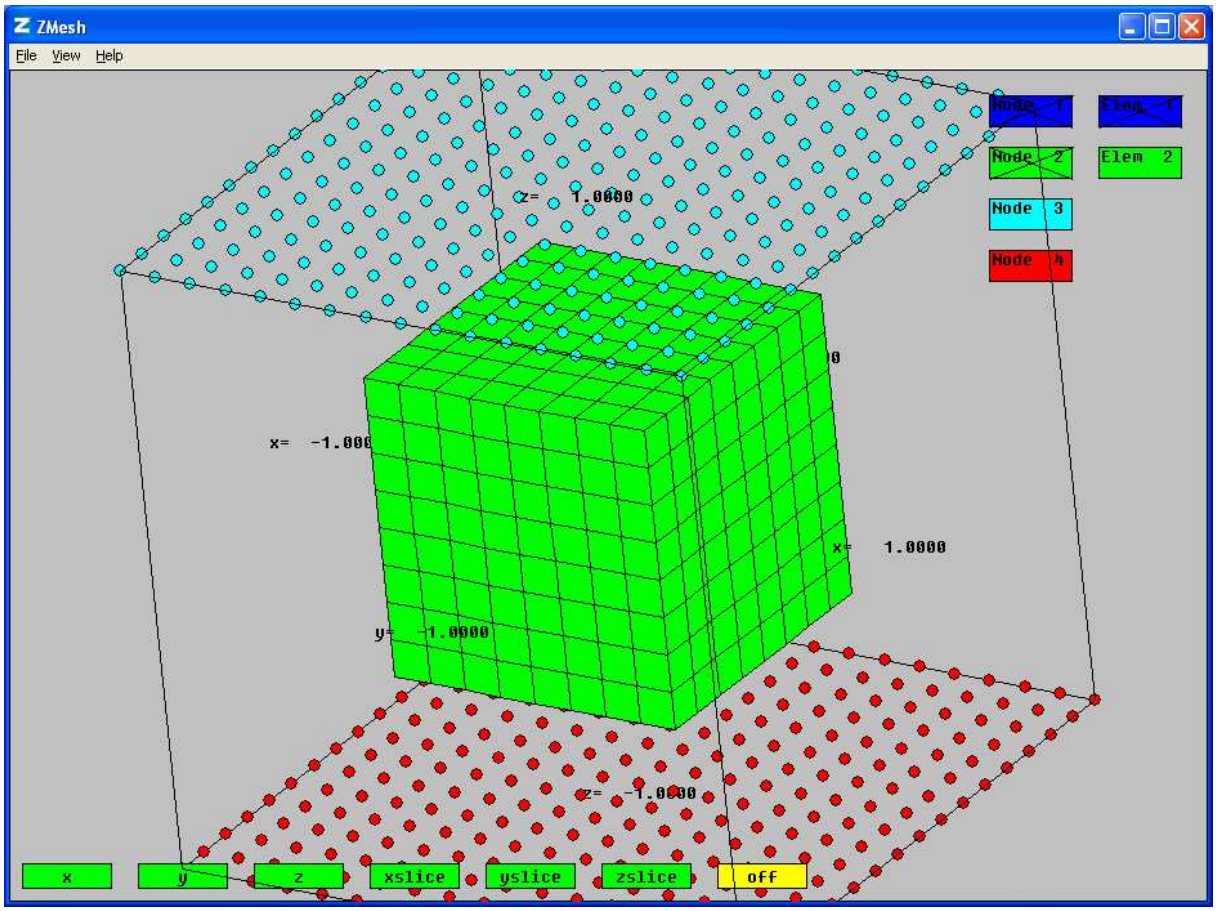


Fig 1: Piezoelectric simulation consisting of two piezoelectrics (cube in a cube). Region 3 nodes (cyan) are clamped and fixed at 1V. Region 4 nodes (red) are clamped and fixed at 0 V.

We consider a very simple geometry as shown in Fig 1. The outer area is a piezoelectric known as PZT-5H while the inner area is a fictitious material with 10 times greater stiffness and the same piezoelectric constant and permittivity. The top and bottom row of nodes shown in Fig 1 are fixed in terms of all the variables: the displacements are fixed at zero ($u_x = u_y = u_z = 0$) while the potential is set at 1 V (top) and 0 V (bottom). In Zinc, it is, of course, possible to fix only a selection of the variables on each node. Here we will fix all variables. We will set the Zinc matrices **g = q = 0** (the default) which results in a zero-stress, zero-charge boundary condition at the x and y minimum and maximum edges. The simulation is rather similar to that of Example 1 but with extra elastic interactions.

In Chapter 4 of the User Manual, the full C matrix for a multiferroic material is derived resulting in equation (4.36). We won't need all that matrix since we are only solving for a piezoelectric. In fact, the C-matrix is given by the top left portion of that multiferroic matrix

$$\left[\begin{array}{ccccccccc|ccc}
c_{11} & c_{16} & c_{15} & c_{16} & c_{12} & c_{14} & c_{15} & c_{14} & c_{13} & e_{11} & e_{21} & e_{31} \\
c_{61} & c_{66} & c_{65} & c_{66} & c_{62} & c_{64} & c_{65} & c_{64} & c_{63} & e_{16} & e_{26} & e_{36} \\
c_{51} & c_{56} & c_{55} & c_{56} & c_{52} & c_{54} & c_{55} & c_{54} & c_{53} & e_{15} & e_{25} & e_{35} \\
c_{61} & c_{66} & c_{65} & c_{66} & c_{62} & c_{64} & c_{65} & c_{64} & c_{63} & e_{16} & e_{26} & e_{36} \\
c_{21} & c_{26} & c_{25} & c_{26} & c_{22} & c_{24} & c_{25} & c_{24} & c_{23} & e_{12} & e_{22} & e_{32} \\
c_{41} & c_{46} & c_{45} & c_{46} & c_{42} & c_{44} & c_{45} & c_{44} & c_{43} & e_{14} & e_{24} & e_{34} \\
c_{51} & c_{56} & c_{55} & c_{56} & c_{52} & c_{54} & c_{55} & c_{54} & c_{53} & e_{15} & e_{25} & e_{35} \\
c_{41} & c_{46} & c_{45} & c_{46} & c_{42} & c_{44} & c_{45} & c_{44} & c_{43} & e_{14} & e_{24} & e_{34} \\
c_{31} & c_{36} & c_{35} & c_{36} & c_{32} & c_{34} & c_{35} & c_{34} & c_{33} & e_{13} & e_{23} & e_{33} \\
\hline
e_{11} & e_{16} & e_{15} & e_{16} & e_{12} & e_{14} & e_{15} & e_{14} & e_{13} & -\kappa_{11} & -\kappa_{12} & -\kappa_{13} \\
e_{21} & e_{26} & e_{25} & e_{26} & e_{22} & e_{24} & e_{25} & e_{24} & e_{23} & -\kappa_{21} & -\kappa_{22} & -\kappa_{23} \\
e_{31} & e_{36} & e_{35} & e_{36} & e_{32} & e_{34} & e_{35} & e_{34} & e_{33} & -\kappa_{31} & -\kappa_{32} & -\kappa_{33}
\end{array}\right]$$

where c_{11} etc are the elastic stiffnesses (N/m2), e_{11} etc are the piezoelectric constants(C/m2), κ_{11} etc are the permittivities (F/m). The upper left block of this matrix gives the elastic behaviour (the most complicated part of the simulation); the lower right block gives the dielectric behaviour and is essentially the same as the matrix we derived in Example 1, except we allow for anisotropy. The off diagonal blocks specify the elastic-electric coupling. See User Manual, chapter 4 for the full derivation of this matrix.

We will use "**C**" for the Zinc matrix and "**c**" for elastic stiffnesses. Although the two are related, it's important not to get confused!

In our problem, all the other matrices can be set to zero as we have no body/surface forces or charges. If these were required the **f** matrix would be used for body forces and charges while the **g** matrix would be used for surface forces and charges. In the case of steady state, elastic vibration, the **a** matrix can be used to encode the material densities and vibration frequency, in fact the a vector would be given as

$a = diag(-\rho\omega^2, -\rho\omega^2, -\rho\omega^2, 0)$

This will not be needed here, however, since we are modelling a static situation.

In the Zinc input file, piezo.zin, it is necessary to enter the C matrix in its component form. The equivalence is shown in Appendix A, equation A3. Thus, for example

$\overline{C}_{1111} = \overline{C}^{11}_{11} = c_{11}$
$\overline{C}_{1112} = \overline{C}^{11}_{12} = c_{16}$
$\overline{C}_{1113} = \overline{C}^{11}_{13} = c_{15}$
Etc…

Which is entered using in the ZIN file using

```
region 1 elements 144 values C
    1    1    1    1  = 0.12600E+12    ! c11
    1    1    1    2  = 0.00000E+00    ! c16
    1    1    1    3  = 0.00000E+00    ! c15
:
etc
```

Note that, for example, that c_{16} is just a conventional way to specify the stiffness tensor component c_{1112} (both in N/m^2). Thus the Zinc C matrix components corresponding to elasticity are simply equal to stiffness tensor components. The relation between tensor and matrix components for elastic stiffness is

tensor	11	22	33	23,32	13,31	12,21
matrix	1	2	3	4	5	6

which is the standard used in elasticity textbooks.

The Zinc input file piezo.zin is pretty long for this problem and is not all shown here (see examples/piezo/piezo.zin). In fact, we generated it automatically using a computer program. Thus, as can be seen above, we have set all the stiffness components even those which are zero. Also, we have input the constants directly rather than using constants in the piezo.con file. However, the piezo.con file is used for post processing.

Simulation

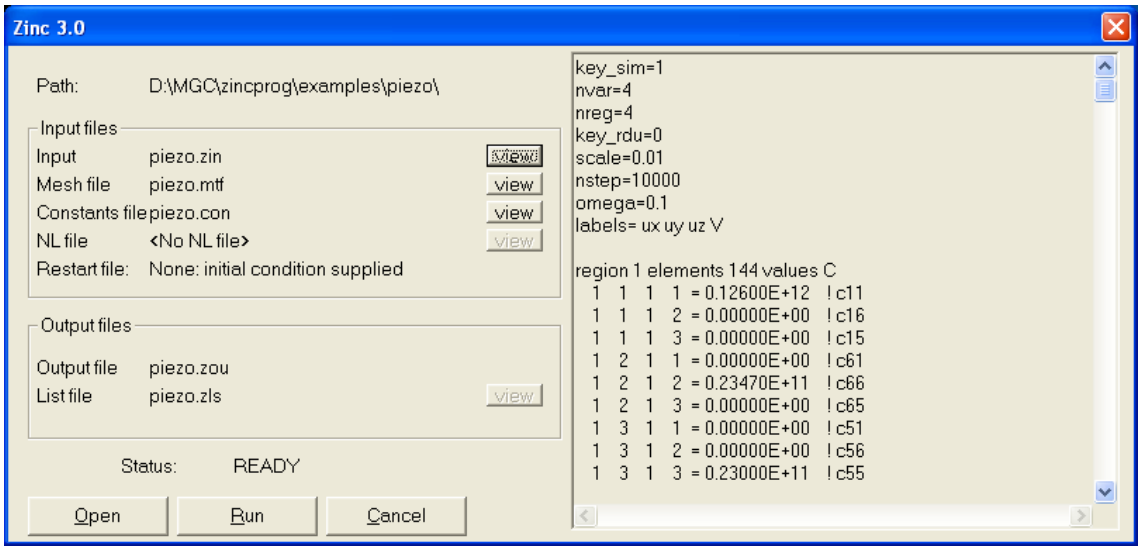


Fig 2: Zinc window ready for simulation.

Having set up the geometry and output the MTF file, we are now ready to run the simulation as shown in Fig 2.

Having run the simulation, we can take a look at the output using Zmesh, Fig 3.

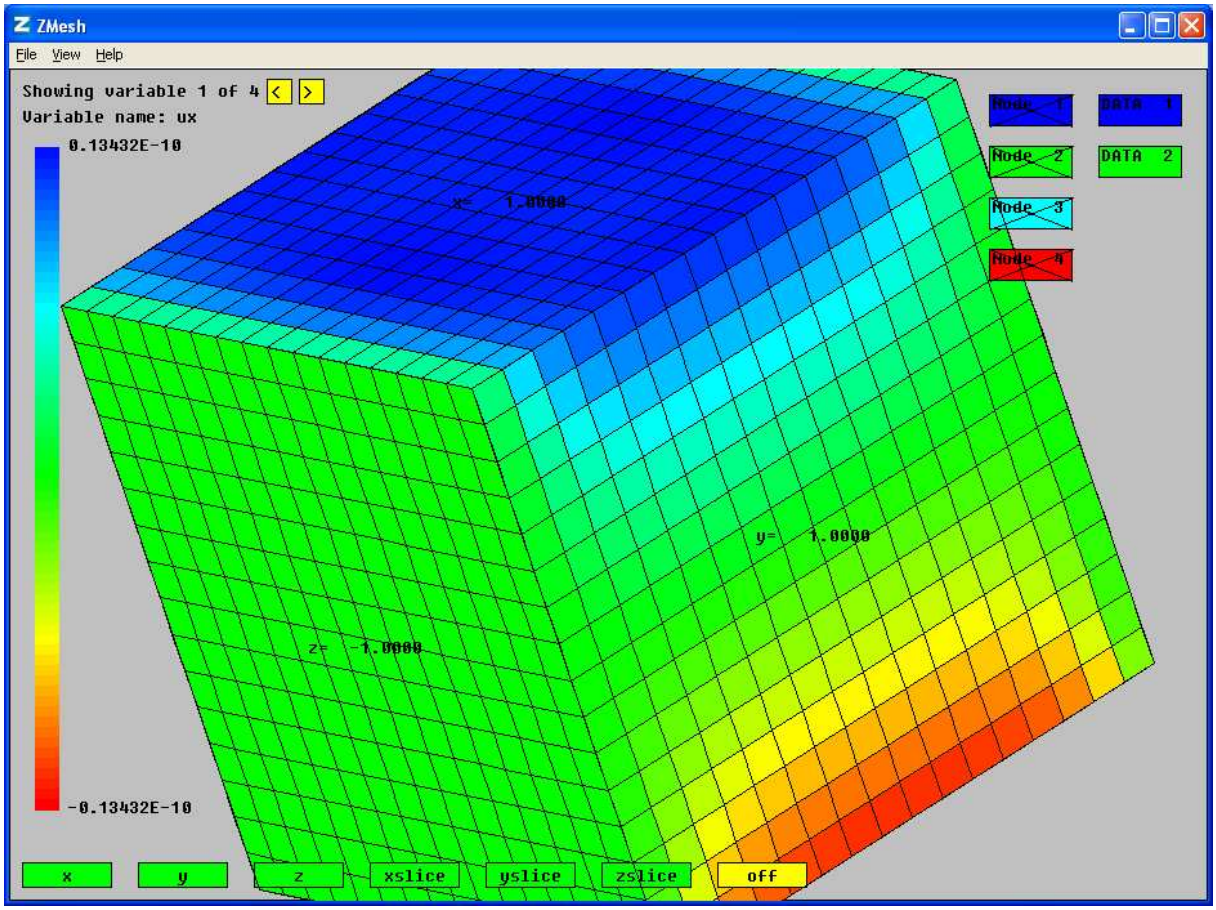
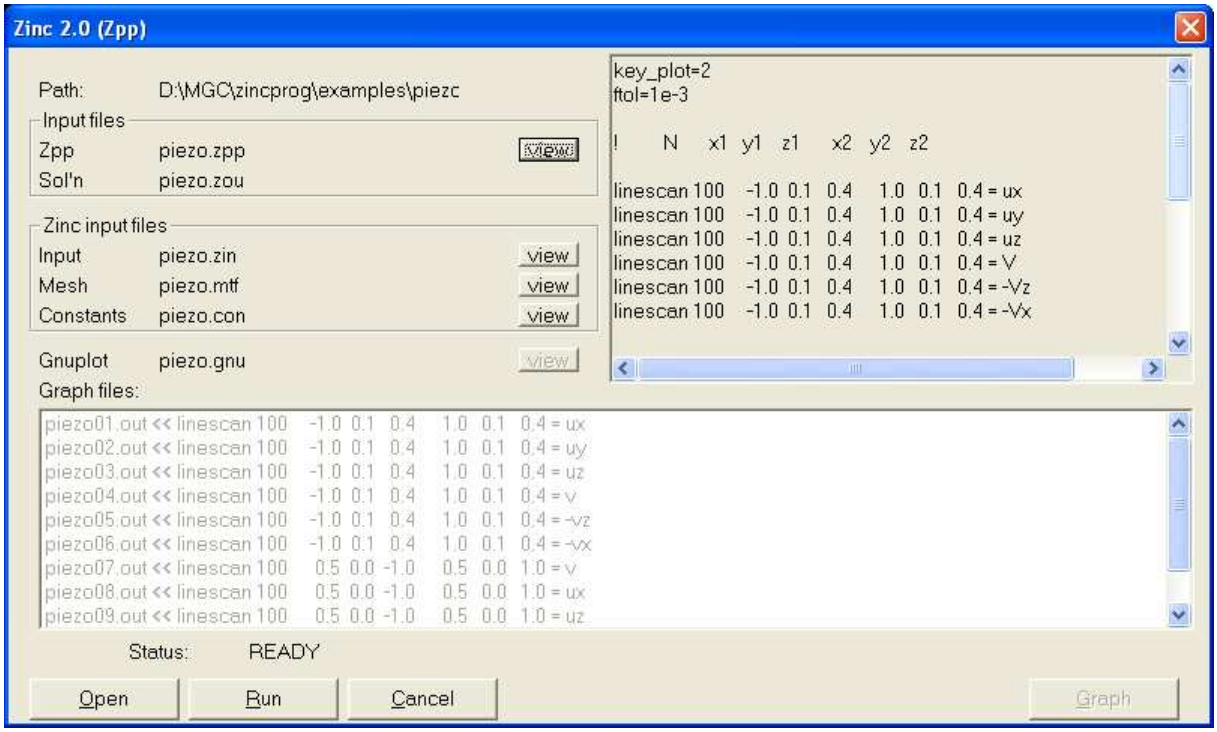


Fig 3: Showing the output using Zmesh. The Yellow < and > buttons can be used to cycle between the four variables solved for (u_x, u_y, u_z, V).

WE now run Zpp to output scans through the data, Fig 4


Zinc 2.0 (Zpp)
Path: D:\MGC\zincprog\examples\piezc
Input files
Zpp piezo.zpp View
Sol'n piezo.zou
Zinc input files
Input piezo.zin view
Mesh piezo.mtf view
Constants piezo.con view
Gnuplot piezo.gnu view
Graph files:
key_plot=2
ftol=1e-3
! N x1 y1 z1 x2 y2 z2
linescan 100 -1.0 0.1 0.4 1.0 0.1 0.4 = ux
linescan 100 -1.0 0.1 0.4 1.0 0.1 0.4 = uy
linescan 100 -1.0 0.1 0.4 1.0 0.1 0.4 = uz
linescan 100 -1.0 0.1 0.4 1.0 0.1 0.4 = V
linescan 100 -1.0 0.1 0.4 1.0 0.1 0.4 = -Vz
linescan 100 -1.0 0.1 0.4 1.0 0.1 0.4 = -Vx
piezo01.out << linescan 100 -1.0 0.1 0.4 1.0 0.1 0.4 = ux
piezo02.out << linescan 100 -1.0 0.1 0.4 1.0 0.1 0.4 = uy
piezo03.out << linescan 100 -1.0 0.1 0.4 1.0 0.1 0.4 = uz
piezo04.out << linescan 100 -1.0 0.1 0.4 1.0 0.1 0.4 = v
piezo05.out << linescan 100 -1.0 0.1 0.4 1.0 0.1 0.4 = -vz
piezo06.out << linescan 100 -1.0 0.1 0.4 1.0 0.1 0.4 = -vx
piezo07.out << linescan 100 0.5 0.0 -1.0 0.5 0.0 1.0 = v
piezo08.out << linescan 100 0.5 0.0 -1.0 0.5 0.0 1.0 = ux
piezo09.out << linescan 100 0.5 0.0 -1.0 0.5 0.0 1.0 = uz
Status: READY
Open
Run
Cancel
Graph
Fig 4: Post Processing with ZPP.

Let's look at the piezo.zpp input file here:

```
key_plot=2
ftol=1e-3

!          N       x1   y1    z1       x2    y2    z2

linescan 100     -1.0  0.1   0.4      1.0   0.1   0.4 = ux
linescan 100     -1.0  0.1   0.4      1.0   0.1   0.4 = uy
linescan 100     -1.0  0.1   0.4      1.0   0.1   0.4 = uz
linescan 100     -1.0  0.1   0.4      1.0   0.1   0.4 = V
linescan 100     -1.0  0.1   0.4      1.0   0.1   0.4 = -Vz
linescan 100     -1.0  0.1   0.4      1.0   0.1   0.4 = -Vx

linescan 100      0.5  0.0  -1.0      0.5   0.0   1.0 = V
linescan 100      0.5  0.0  -1.0      0.5   0.0   1.0 = ux
linescan 100      0.5  0.0  -1.0      0.5   0.0   1.0 = uz


linescan 100     -1.0  0.1   0.4      1.0   0.1   0.4 = &
{1=c11,2=c11inc}*uxx + {1=c12,2=c12inc}*uyy + {1=c13,2=c13inc}*uzz + &
{1=e31,2=e31inc}*Vz

linescan 100     -1.0  0.1   0.4      1.0   0.1   0.4 = &
{1=c12,2=c12inc}*uxx + {1=c22,2=c22inc}*uyy + {1=c23,2=c23inc}*uzz + &
{1=e32,2=e32inc}*Vz
```

The first batch of scans are along x and plot the raw output and also the electric fields. Recall that Vz means dV/dz, so –Vz is the electric field, z-component, Ez.

The next batch of line scans are along z, between the two electrode plates.

The final two scans are stress plots, specifically σ1=σxx and σ2=σyy. Let us consider the constitutive law of piezoelectrics.

$$\begin{pmatrix}\sigma_1\\ \sigma_2\\ \sigma_3\\ \sigma_4\\ \sigma_5\\ \sigma_6\end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} & & & \\ c_{21} & c_{22} & c_{23} & & & \\ c_{31} & c_{32} & c_{33} & & & \\ & & & c_{44} & & \\ & & & & c_{55} & \\ & & & & & c_{66}\end{pmatrix}\begin{pmatrix}\varepsilon_1\\ \varepsilon_2\\ \varepsilon_3\\ \varepsilon_4\\ \varepsilon_5\\ \varepsilon_6\end{pmatrix} - \begin{pmatrix} & & e_{31}\\ & & e_{32}\\ & & e_{33}\\ & e_{24} & \\ e_{15} & & \end{pmatrix}\begin{pmatrix}E_1\\ E_2\\ E_3\end{pmatrix}$$

Multiplying out gives

$$\sigma_1 = c_{11}\varepsilon_1 + c_{12}\varepsilon_2 + c_{13}\varepsilon_3 - e_{31}E_3$$
$$\sigma_2 = c_{21}\varepsilon_1 + c_{22}\varepsilon_2 + c_{23}\varepsilon_3 - e_{32}E_3$$

which are precisely the formulas used in the last two linescans. The expression

```
{1=c12,2=c12inc}
```

means “take c12 when we are in region 1 and c12inc (inclusion) when we are in region 2”. Note that u_{xx} is simply du_x/dx which is the necessary ε_1 strain etc. The last two linescans make use of constants in piezo.con:

```
eps0=8.854e-12

c11=12.6e10
c33=11.7e10
c44=2.3e10
c12=7.95e10
c13=8.41e10
c66=2.347e10

e31=-6.5
e33=23.3
e15=17

k11=1700*eps0
k33=1470*eps0

c22=c11
c23=c13
c55=c44

e32=e31
e24=e15

k22=k11

! ------------------------------------------------------------------------

c11inc=12.6e11
c33inc=11.7e11
c44inc=2.3e11
```

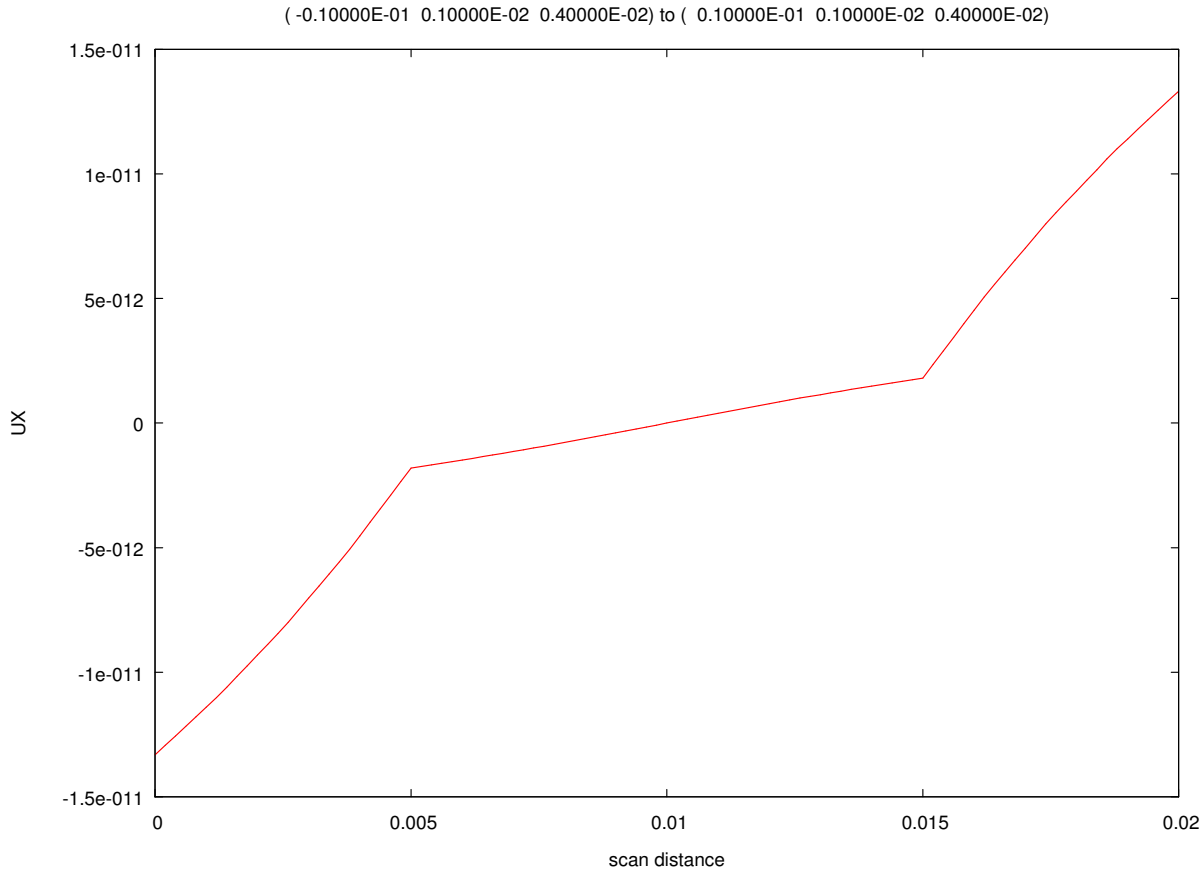
```
c12inc=7.95e11
c13inc=8.41e11
c66inc=2.347e11

e31inc=-6.5
e33inc=23.3
e15inc=17

k11inc=1700*eps0
k33inc=1470*eps0

c22inc=c11inc
c23inc=c13inc
c55inc=c44inc

e32inc=e31inc
e24inc=e15inc

k22inc=k11inc
```

Note that parameters can be defined in terms of *previously-specified* parameters. Here, we use this to enforce the symmetry equivalence of certain coefficients (assuming the piezoelectric has 6mm symmetry and is poled along the z-direction).

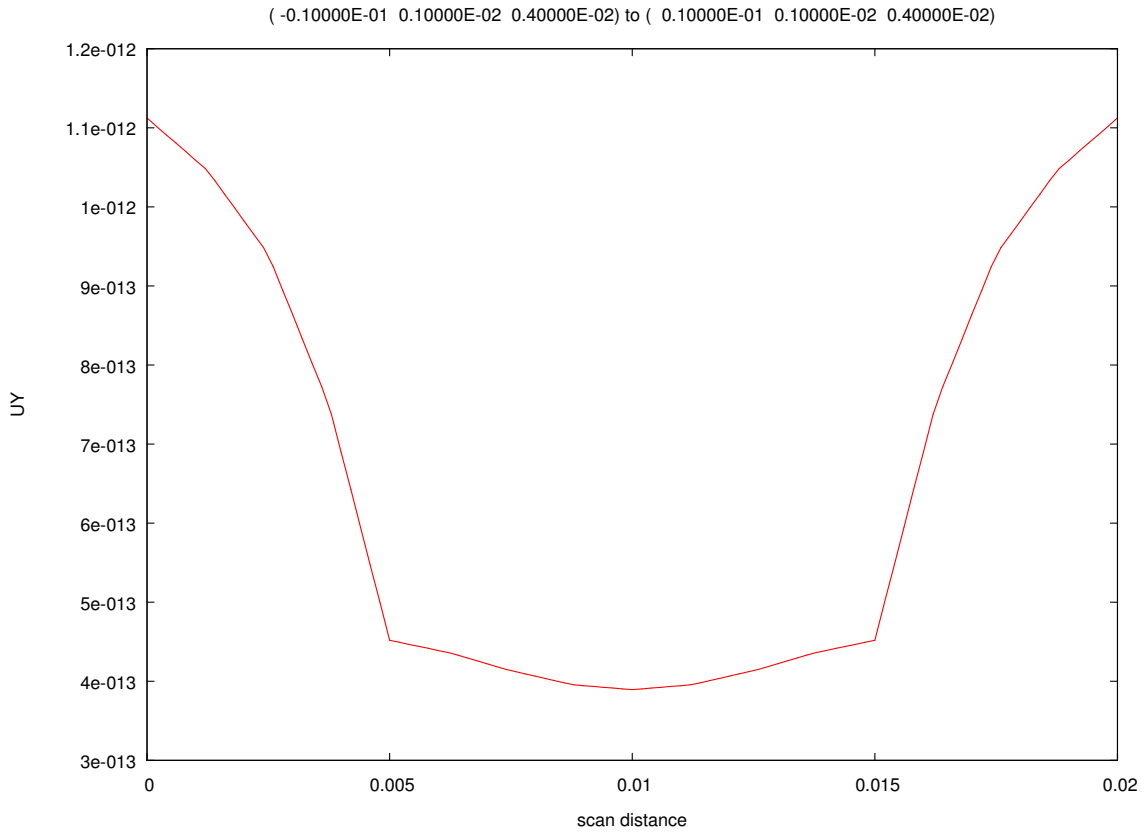
Results

The following are the enhanced metafiles (EMF) supplied with the Zinc distribution for this example (piezo01.emf, piezo02.emf etc). You should check you get the same results!

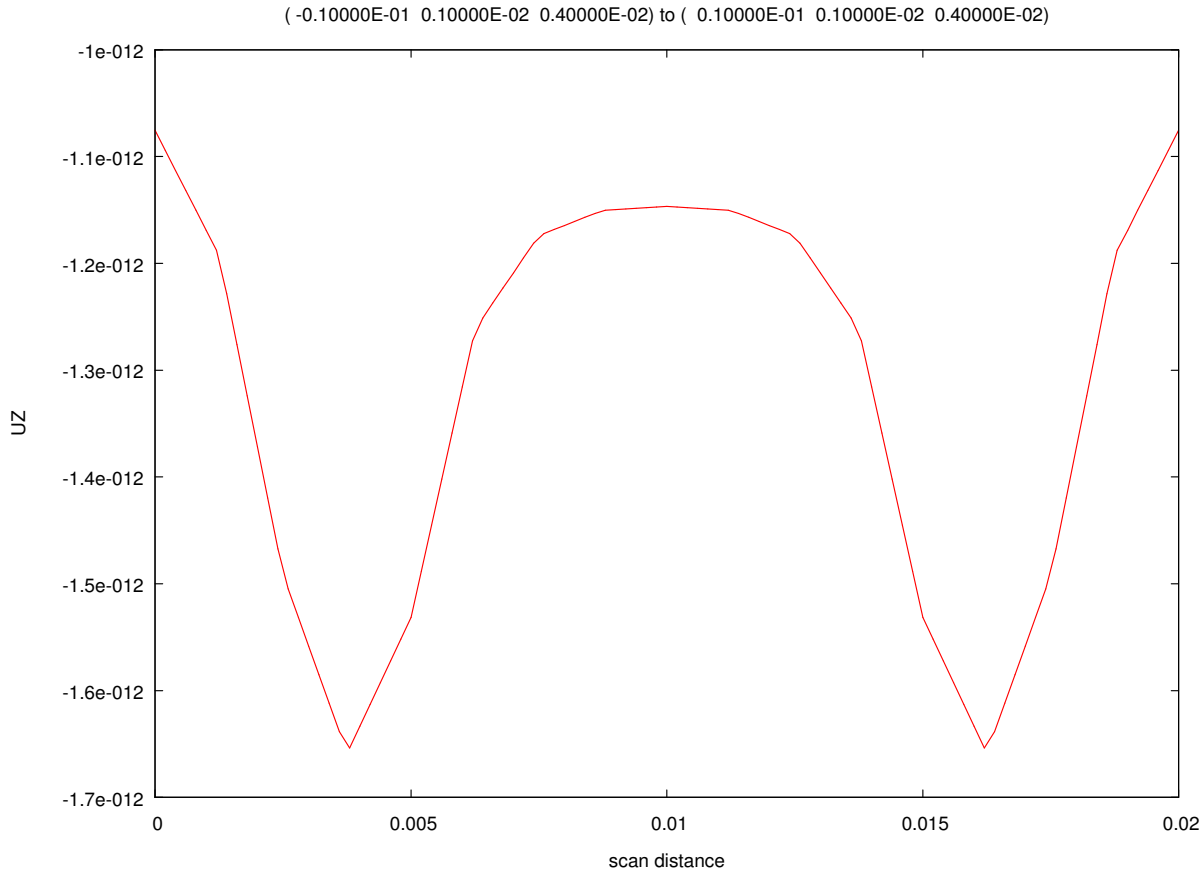
u_x variation x-scan



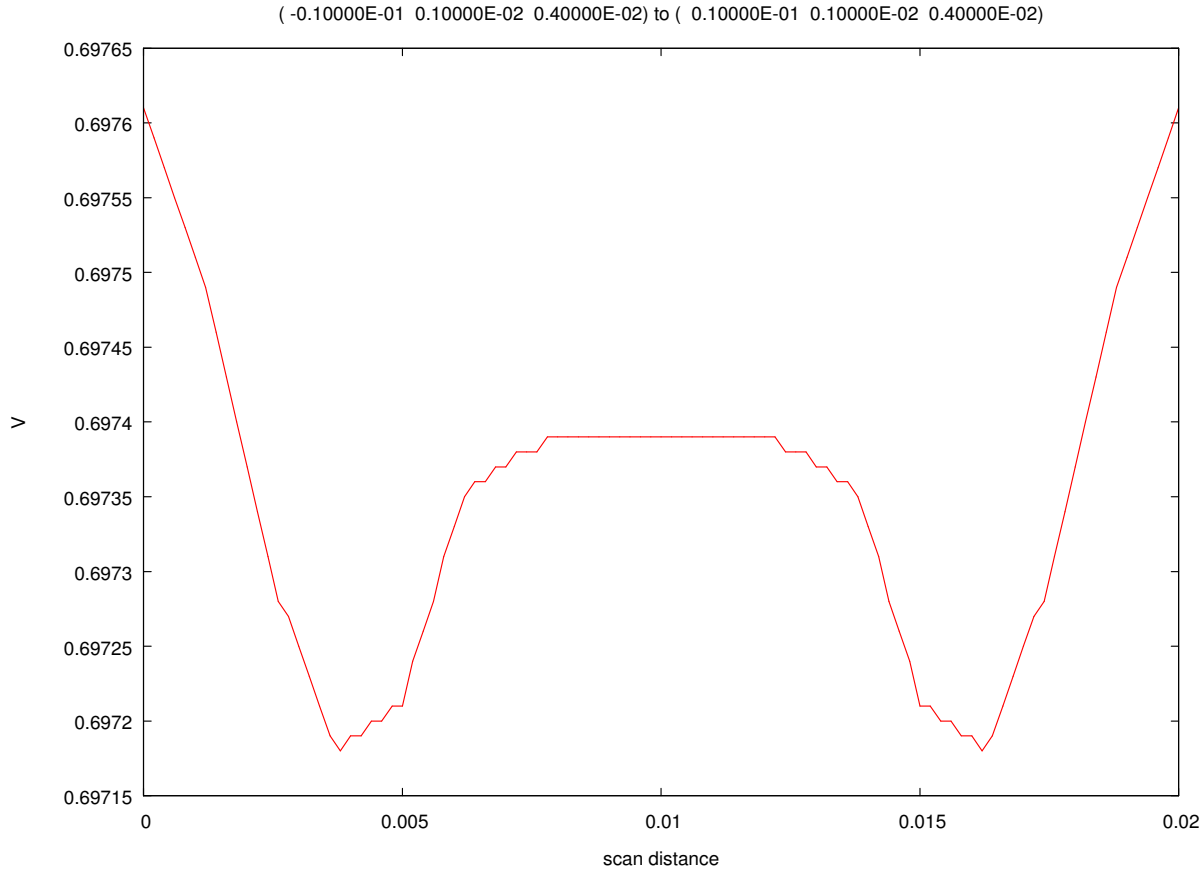
u_y variation x-scan



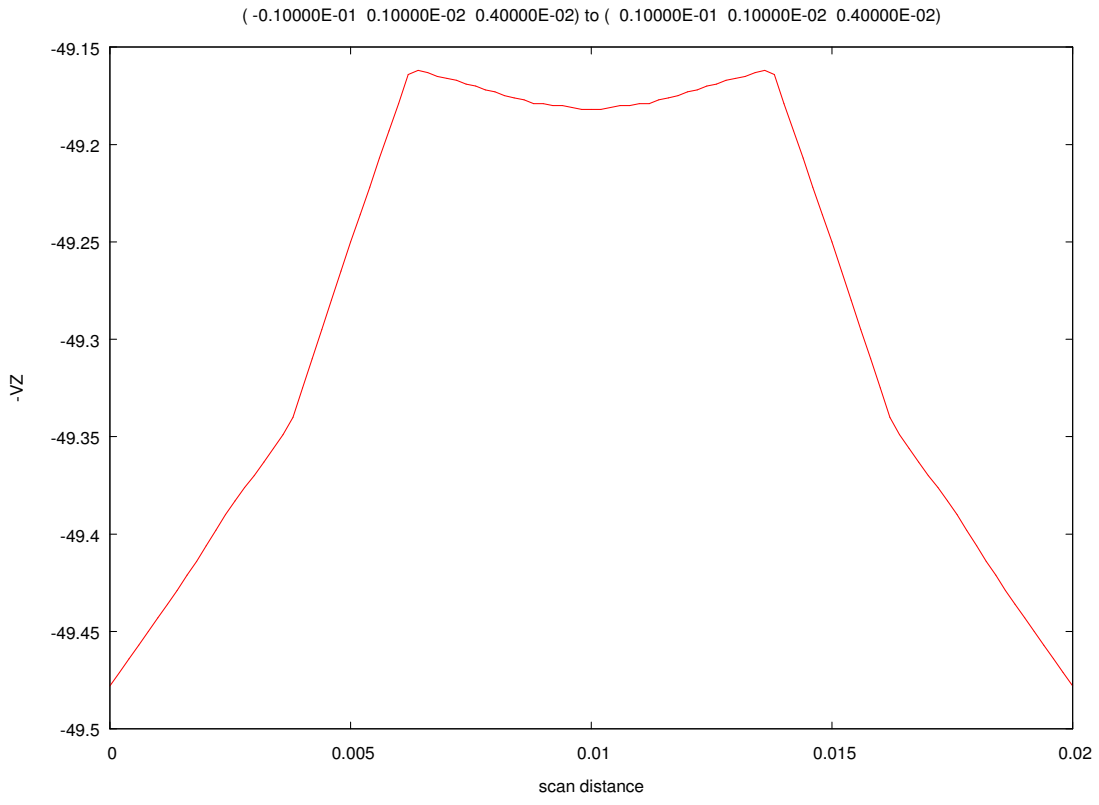
u_z variation x-scan



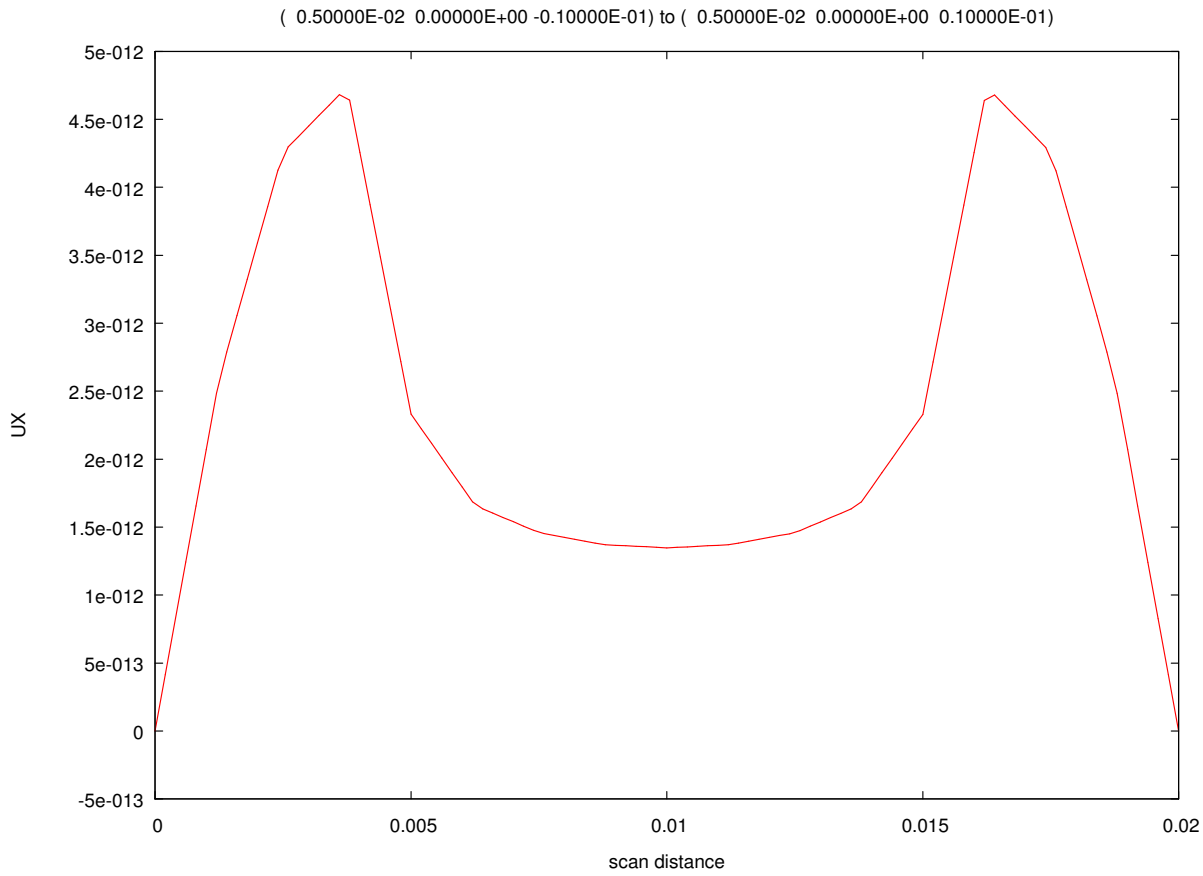
Voltage variation x-scan



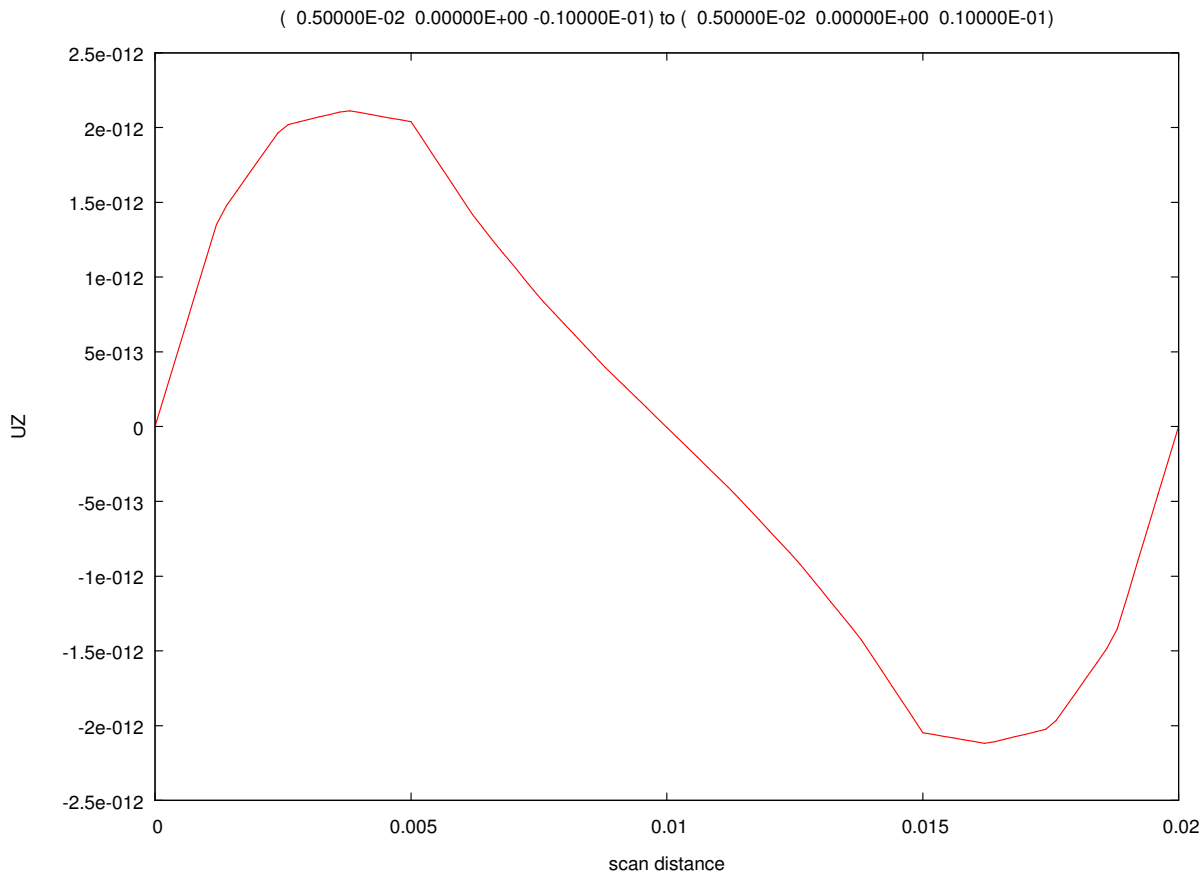
E_z variation x-scan



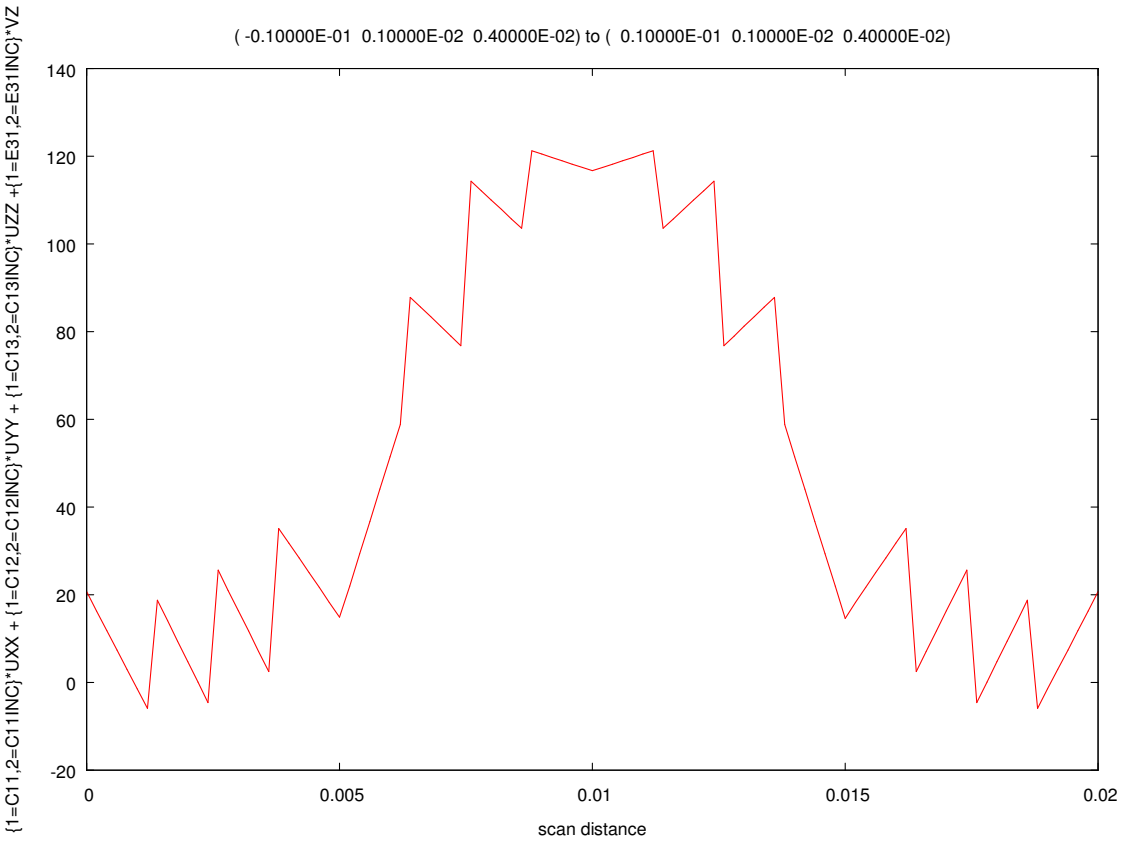
u_x variation z-scan



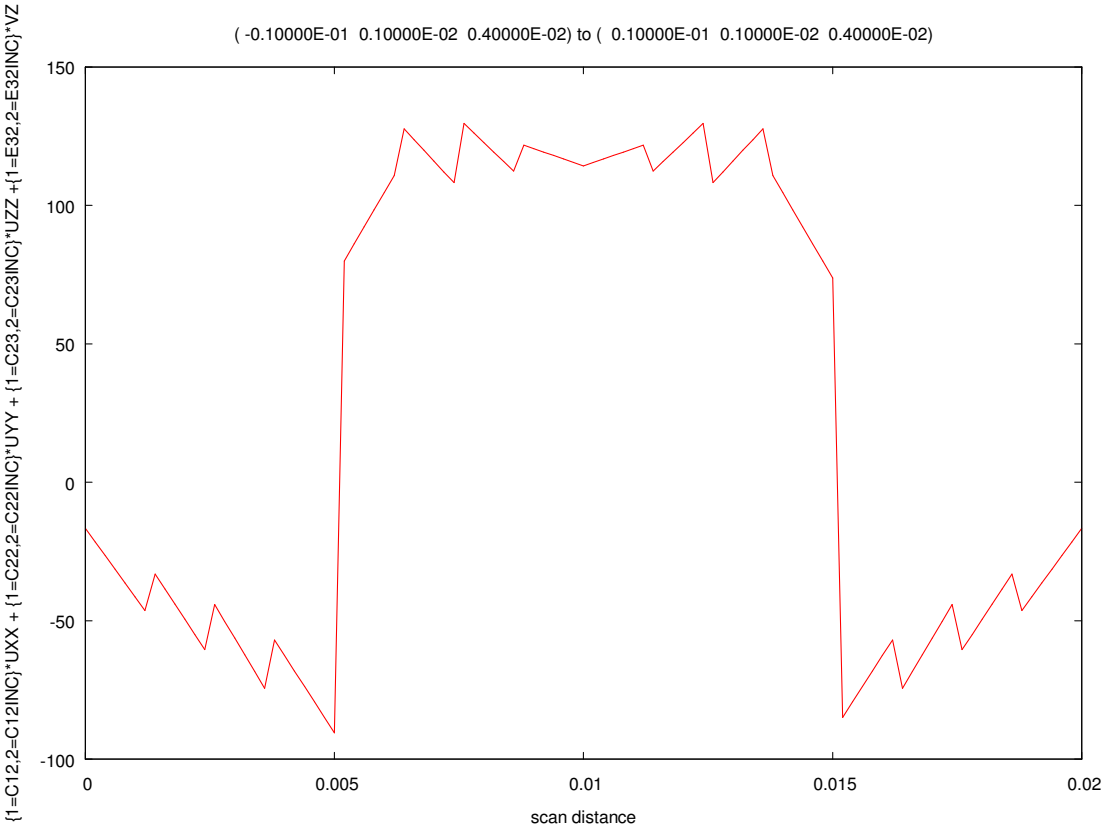
u_z variation z-scan



σ_{xx} x-scan



σ_{yy} x-scan

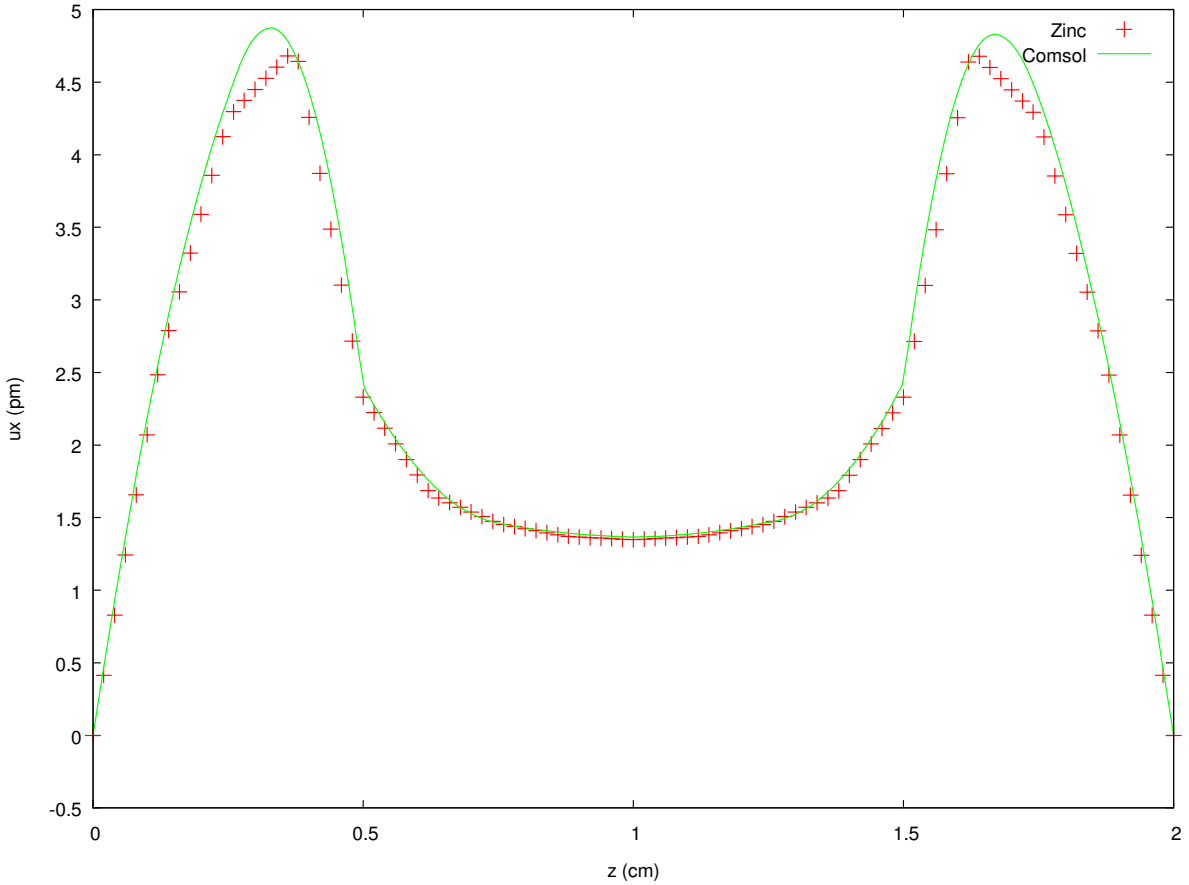


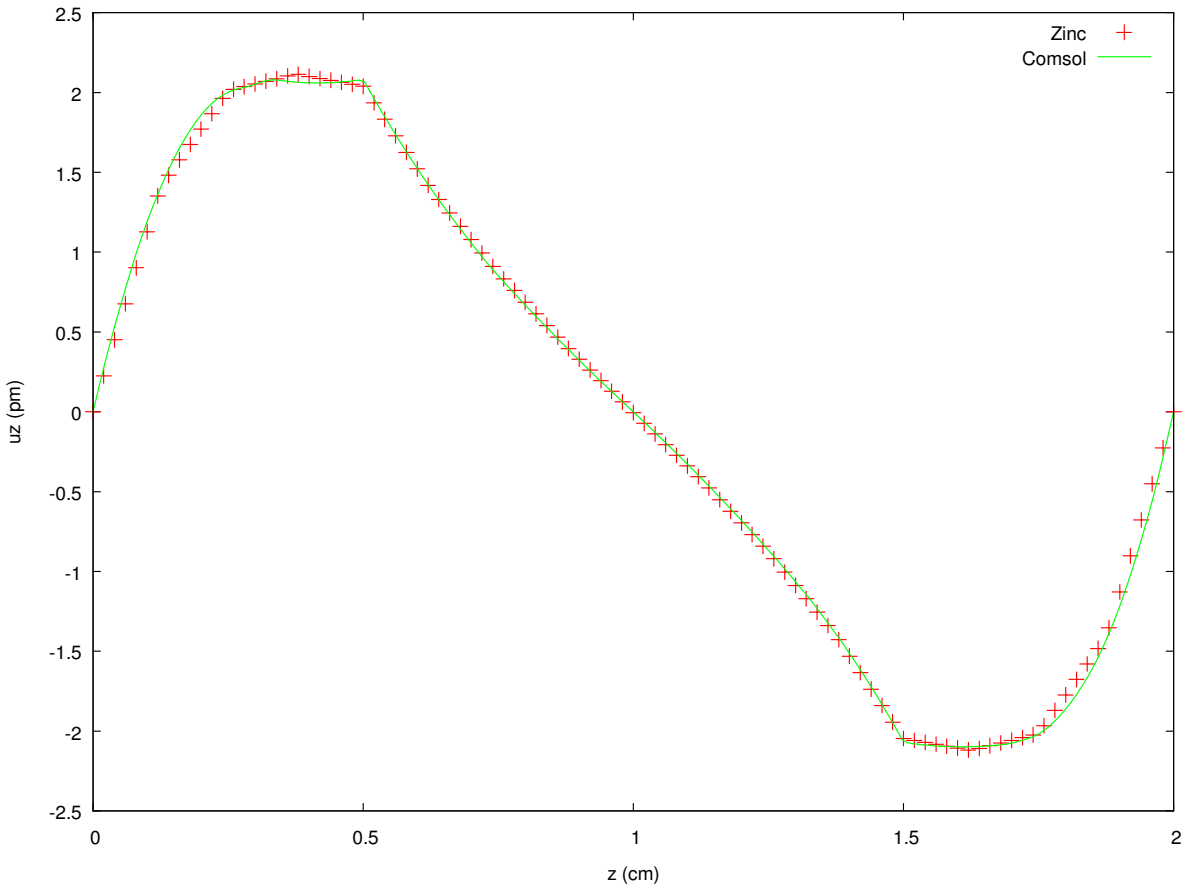
Comparison graphs

To check solution accuracy, it is often a good idea to compare simulation results between different FE solvers. We have therefore compared $u_x(z)$ and $u_z(z)$ with Comsol (another finite element solver). The Comsol results are in files ux.txt and uz.txt in the Zinc distribution. The following gnuplot script file was written to perform the comparison.

```
set term emf

set output 'uzcomp.emf'
set xlabel 'z (cm)'
set ylabel 'uz (pm)'
plot 'piezo09.out' u ($1*100):($5*1e12) t 'Zinc','uz.txt' i 0 u 1:($2*1e12)
t 'Comsol' w l

set output 'uxcomp.emf'
set xlabel 'z (cm)'
set ylabel 'ux (pm)'
plot 'piezo08.out' u ($1*100):($5*1e12) t 'Zinc','ux.txt' i 0 u 1:($2*1e12)
t 'Comsol' w l
```

This results in the following graphs which show good agreement.





Also in the directory

Piezoslap.zin: Same simulation but using the Incomplete LU factorisation GMRES method.

Example 3: Fuel cell

Directory: examples/fuelcell

Cathode.zin: self contained simulation without programming
Cathodetoken.zin: simulation with DLL written in Fortran
Cathodetokenc.zin: simulation with DLL written in C.

Introduction

We now consider a multiphysics, non-linear example, the fuel cell. Fuel cells are ideal cases for general purpose multiphysics solvers like Zinc since the precise set of equations and boundary conditions is under constant debate. It is often necessary to solve a set of equations only recently derived which cannot be done with traditional FE packages that have built in modules corresponding to common problems. Zinc allows you to write down a new set of equations and immediately solve them without having to wait for developers to explicitly add the new equation sets in. This is precisely the approach we took in a recent fuel cell project at NPL. We derived the equations ourselves from first principles and them implemented them in Zinc. The full derivation of the fuel cell equations is given in NPL report MAT (RES) 108. Here we will briefly summarise the derivation and show the working equation set that needs to be implemented.

The fundamental equation for diffusion of n species is

$$\frac{\partial(\rho\omega_k)}{\partial t} + \nabla \cdot \mathbf{j}_k + \rho\mathbf{u} \cdot \nabla\omega_k = R_k, \; k = 1\ldots n-1 \qquad (1)$$

where ρ denotes the density (kg/m^3), $\omega_k = \rho_k / \rho$ is the mass fraction of species k, **u** is the mass-based mean velocity vector (m/s) and R_k is the reaction rate (kg/(m^3·s)). Also, $\mathbf{j}_k = \rho_k(\mathbf{u}_k - \mathbf{u})$ describes the diffusion-driven transport where $\mathbf{u}_k$ is the mean velocity of the kth species

The flux $\mathbf{j}_k$ is given by

$$\mathbf{j}_k = -D_k^T \frac{\nabla T}{T} - \rho\omega_k \sum_{j=1}^{n} D_{kj}\mathbf{d}_j, \quad k = 1\ldots n, \qquad (2)$$

where the "driving force" vector **d** is given by

$$\mathbf{d}_j = \left\{ \nabla x_j + (x_j - \omega_j)\frac{\nabla p}{p} \right\}, \quad j = 1\ldots n, \qquad (3)$$

where $x_j = c_j / c$ is the molar fraction of species j and p is the pressure (Pa), and where c_j is the concentration (moles/m^3) of the jth species and c is the mean molar concentration defined by

$$c = \sum_{k=1}^{n} c_k$$

Thus, the driving force for diffusion comes from two parts, the variation in species concentration x_j and the variation of pressure, p.

The molar and mass fractions satisfy the relations

$$\sum_{k=1}^{n} x_k = 1, \qquad \sum_{k=1}^{n} \omega_k = 1,$$

and can be expressed in terms of each other using

$$\omega_j = \frac{m_j}{m} x_j$$

with

$$m = \sum_{j=1}^{n} m_j x_j$$

The total density ρ, in (1), can be written as

$$\rho = \frac{p}{RT}\sum_{j=1}^{n} m_j x_j$$

We now make several assumptions to simplify the problem. Firstly, we assume steady state so that d/dt=0. Next, we assume no temperature variation for that $\nabla T = 0$. Finally, we assume reaction rates, R_k of zero. With these assumptions, substituting (3) into (2) and then (2) into (1) gives:

$$\nabla \cdot \left\{ \rho \omega_k \sum_j D_{kj} \left[\nabla x_j + (x_j - \omega_j)\frac{\nabla p}{p} \right] \right\} + \rho \mathbf{u} \cdot \nabla \omega_k = 0 \qquad k=1,\ldots n\text{-}1 \qquad (4)$$

In our problem we will consider a cathode simulation as shown in Fig 1. We will take n=3 species namely O_2, H_2O and N_2. Since we are dealing with mass fractions (which sum to unity we need only consider the first two species. Thus there are three variables to be solved: (x_{O2}, x_{H2O}, p)

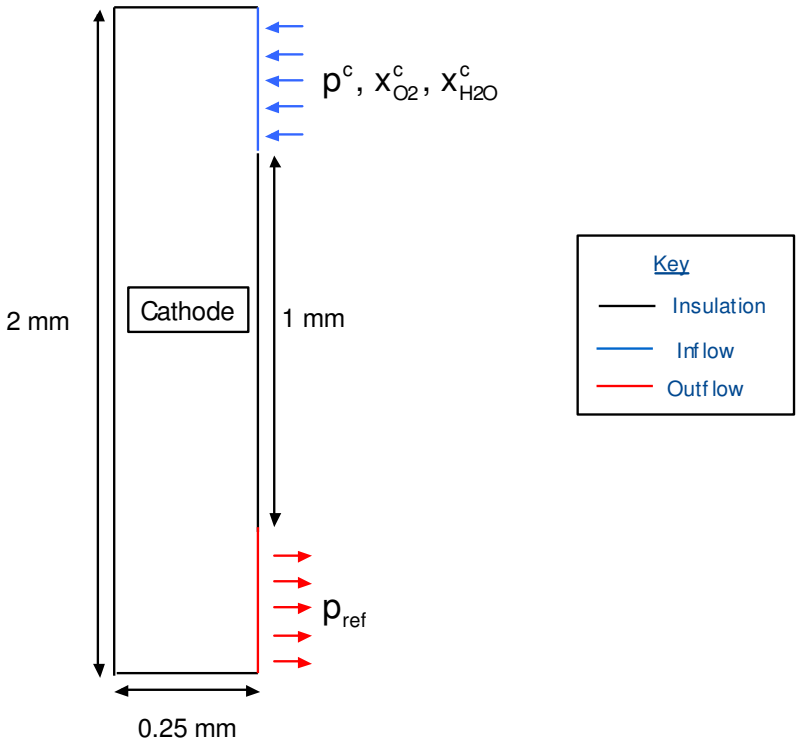


Figure 1: Geometry and boundary conditions for the cathode model

The D coefficients in (4) are not constants but themselves depend on the mass fractions as

$$D_{11} = \frac{\dfrac{(\omega_2+\omega_3)^2}{x_1 \tilde{D}_{23}} + \dfrac{\omega_2^2}{x_2 \tilde{D}_{13}} + \dfrac{\omega_3^2}{x_3 \tilde{D}_{12}}}{\dfrac{x_1}{\tilde{D}_{12}\tilde{D}_{13}} + \dfrac{x_2}{\tilde{D}_{12}\tilde{D}_{23}} + \dfrac{x_3}{\tilde{D}_{13}\tilde{D}_{23}}},$$

$$D_{22} = \frac{\dfrac{(\omega_1+\omega_3)^2}{x_2\tilde{D}_{13}} + \dfrac{\omega_1^2}{x_1\tilde{D}_{23}} + \dfrac{\omega_3^2}{x_3\tilde{D}_{12}}}{\dfrac{x_1}{\tilde{D}_{12}\tilde{D}_{13}} + \dfrac{x_2}{\tilde{D}_{12}\tilde{D}_{23}} + \dfrac{x_3}{\tilde{D}_{13}\tilde{D}_{23}}},$$

$$D_{33} = \frac{\dfrac{(\omega_1+\omega_2)^2}{x_3\tilde{D}_{12}} + \dfrac{\omega_1^2}{x_1\tilde{D}_{23}} + \dfrac{\omega_2^2}{x_2\tilde{D}_{13}}}{\dfrac{x_1}{\tilde{D}_{12}\tilde{D}_{13}} + \dfrac{x_2}{\tilde{D}_{12}\tilde{D}_{23}} + \dfrac{x_3}{\tilde{D}_{13}\tilde{D}_{23}}},$$

$$D_{12} = \frac{\dfrac{\omega_1(\omega_2+\omega_3)}{x_1\tilde{D}_{23}} + \dfrac{\omega_2(\omega_1+\omega_3)}{x_2\tilde{D}_{13}} - \dfrac{\omega_3^2}{x_3\tilde{D}_{12}}}{\dfrac{x_1}{\tilde{D}_{12}\tilde{D}_{13}} + \dfrac{x_2}{\tilde{D}_{12}\tilde{D}_{23}} + \dfrac{x_3}{\tilde{D}_{13}\tilde{D}_{23}}},$$

$$D_{13} = \frac{\dfrac{\omega_1(\omega_2+\omega_3)}{x_1\tilde{D}_{23}} + \dfrac{\omega_3(\omega_1+\omega_2)}{x_3\tilde{D}_{12}} - \dfrac{\omega_2^2}{x_2\tilde{D}_{13}}}{\dfrac{x_1}{\tilde{D}_{12}\tilde{D}_{13}} + \dfrac{x_2}{\tilde{D}_{12}\tilde{D}_{23}} + \dfrac{x_3}{\tilde{D}_{13}\tilde{D}_{23}}},$$

$$D_{23} = \frac{\dfrac{\omega_2(\omega_1+\omega_3)}{x_2\tilde{D}_{13}} + \dfrac{\omega_3(\omega_1+\omega_2)}{x_3\tilde{D}_{12}} - \dfrac{\omega_1^2}{x_1\tilde{D}_{23}}}{\dfrac{x_1}{\tilde{D}_{12}\tilde{D}_{13}} + \dfrac{x_2}{\tilde{D}_{12}\tilde{D}_{23}} + \dfrac{x_3}{\tilde{D}_{13}\tilde{D}_{23}}}.$$

The D coefficient matrix is symmetrical. Tables 1-4 summarise the full set of material properties used in the simulation. These are present in the constants file (cathode.con, cathodetoken.con).

Table 1: Constants for the electrode flow models.

Name	Value	Description	Units
R	8.314	Gas constant	J/K·mol
T	353	Temperature	K
KH2	3.9×10^{4}	Henry's constant for H2 dissolved in H2O	Pa·m3/mol
KO2	3.2×10^{4}	Henry's constant for O2 dissolved in H2O	Pa·m3/mol
mO2	32.0×10^{-3}	Molar mass of O2	kg/mol
mH2	2.0×10^{-3}	Molar mass of H2	kg/mol
mH2O	18.0×10^{-3}	Molar mass of H2O	kg/mol
mN2	28.0×10^{-3}	Molar mass of N2	kg/mol
κ	1.0×10^{-13}	Permeability	M^2
η	2.1×10^{-5}	Fluid/gas viscosity	Pa·s
P_{ref}	1.013×10^{5}	Reference pressure	Pa
ε	0.4	Macroscopic porosity	

Table 2: Effective diffusion coefficients.

Name	Value	Description	Units
$\tilde{D}_{O2N2}$	0.0000074	Binary diffusion coefficient for O_2 and N_2	m^2/s
$\tilde{D}_{O2H2O}$	0.0000087	Binary diffusion coefficient for O_2 and H_2O	m^2/s
$\tilde{D}_{H2H2O}$	0.0000285	Binary diffusion coefficient for O_2 and N_2	m^2/s
$\tilde{D}_{N2H2O}$	0.0000080	Binary diffusion coefficient for O_2 and N_2	m^2/s

Table 3: Inlet parameters.

ω^a_{H2}	0.1	Mass fraction of H_2 at anode inlet	-
ω^a_{H2O}	0.9	Mass fraction of H_2O at anode inlet	-
ω^c_{O2}	0.168	Mass fraction of O_2 at cathode inlet	-
ω^c_{H2O}	0.2	Mass fraction of H_2O at cathode inlet	-
ω^c_{N2}	0.632	Mass fraction of N_2 at cathode inlet	-
x^a_{H2}	0.5	Molar fraction of H_2 at anode inlet	-
x^a_{H2O}	0.5	Molar fraction of H_2O at anode inlet	-
x^c_{O2}	0.1348486	Molar fraction of O_2 at cathode inlet	-
x^c_{H2O}	0.2853939	Molar fraction of H_2O at cathode inlet	-
x^c_{N2}	0.5797574	Molar fraction of N_2 at cathode inlet	-
p^a	111430	Pressure at the anode inlet	Pa
p^c	111430	Pressure at the cathode inlet	Pa

Table 4: Initial conditions/parameters.

ω^a_{H2}	0.5	Mass fraction of H_2 in the anode	-
ω^a_{H2O}	0.5	Mass fraction of H_2O in the anode	-
ω^c_{O2}	1/3	Mass fraction of O_2 in the cathode	-
ω^c_{H2O}	1/3	Mass fraction of H_2O in the cathode	-
ω^c_{N2}	1/3	Mass fraction of N_2 in the cathode	-
x^a_{H2}	0.9	Molar fraction of H_2 in the anode	-
x^a_{H2O}	0.1	Molar fraction of H_2O in the anode	-
x^c_{O2}	0.2550607	Molar fraction of O_2 in the cathode	-
x^c_{H2O}	0.4534413	Molar fraction of H_2O in the cathode	-
x^c_{N2}	0.2914980	Molar fraction of N_2 in the cathode	-
p^a	1.013×10^5	Pressure in the anode	Pa
p^c	1.013×10^5	Pressure in the cathode	Pa

Zinc implementation

We now need to see how to implement equation (4) in Zinc. The Zinc PDE specification is

$$\begin{aligned} -\nabla\cdot(\mathbf{C}\nabla\mathbf{u}) + a\mathbf{u} &= f\,, \\ \mathbf{n}\cdot(\mathbf{C}\nabla\mathbf{u}) + q\mathbf{u} &= g. \end{aligned} \tag{5}$$

We will set a=q=g=0 and put non-zero values only in the C and f matrices. Let us split the C matrix into 3 parts

$$\mathbf{C} = \begin{bmatrix} \mathbf{C}_1 \\ \mathbf{C}_2 \\ \mathbf{C}_3 \end{bmatrix}$$

Then, comparison of (5) with (4) shows that

$$\mathbf{C}_1 = \begin{bmatrix} B_{11} & 0 & 0 & B_{12} & 0 & 0 & Q_1 & 0 & 0 \\ 0 & B_{11} & 0 & 0 & B_{12} & 0 & 0 & Q_1 & 0 \\ 0 & 0 & B_{11} & 0 & 0 & B_{12} & 0 & 0 & Q_1 \end{bmatrix}, \tag{6}$$

$$\mathbf{C}_2 = \begin{bmatrix} B_{21} & 0 & 0 & B_{22} & 0 & 0 & Q_2 & 0 & 0 \\ 0 & B_{21} & 0 & 0 & B_{22} & 0 & 0 & Q_2 & 0 \\ 0 & 0 & B_{21} & 0 & 0 & B_{22} & 0 & 0 & Q_2 \end{bmatrix}, \tag{7}$$

$$\mathbf{C}_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & E & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & E & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & E \end{bmatrix}. \tag{8}$$

with

$$B_{kj} = \frac{pm_k x_k}{RT}(D_{kj} - D_{kn}), \quad j, k = 1 \ldots n-1, \tag{9}$$

$$Q_k = \sum_{j=1}^{n-1} A_{kj} + C_k, \quad k = 1 \ldots n-1, \tag{10}$$

$$E = \frac{pm}{RT}\frac{\kappa}{\eta}, \tag{11}$$

with A_k and C_k defined as

$$A_{kj} = \frac{m_k x_k}{RT} x_j \left[D_{kj}\left(1 - \frac{m_j}{m}\right) - D_{kn}\left(1 - \frac{m_n}{m}\right) \right], \qquad (12)$$

$$C_k = D_{kn} \frac{m_k x_k}{RT}\left(1 - \frac{m_n}{m}\right) \qquad (13)$$

This completes the specification of the C matrix. The f-matrix is given by

$$\mathbf{f} = \begin{pmatrix} f_1 \\ f_2 \\ 0 \end{pmatrix},$$

with

$$f_1 = \frac{p\kappa m_1}{\eta RT}\left\{\frac{\partial p}{\partial x}\left(\frac{\partial x_1}{\partial x} - \frac{x_1}{m}\frac{\partial m}{\partial x}\right) + \frac{\partial p}{\partial y}\left(\frac{\partial x_1}{\partial y} - \frac{x_1}{m}\frac{\partial m}{\partial y}\right) + \frac{\partial p}{\partial z}\left(\frac{\partial x_1}{\partial z} - \frac{x_1}{m}\frac{\partial m}{\partial z}\right)\right\},$$

$$f_2 = \frac{p\kappa m_2}{\eta RT}\left\{\frac{\partial p}{\partial x}\left(\frac{\partial x_2}{\partial x} - \frac{x_2}{m}\frac{\partial m}{\partial x}\right) + \frac{\partial p}{\partial y}\left(\frac{\partial x_2}{\partial y} - \frac{x_2}{m}\frac{\partial m}{\partial y}\right) + \frac{\partial p}{\partial z}\left(\frac{\partial x_2}{\partial z} - \frac{x_2}{m}\frac{\partial m}{\partial z}\right)\right\},$$

$$f_3 = 0.$$

From (5), the Neumann boundary conditions that will exist on non fixed surfaces of the simulation are

$$\begin{pmatrix} -\mathbf{n}\cdot\mathbf{j}_1 \\ -\mathbf{n}\cdot\mathbf{j}_2 \end{pmatrix} = 0,$$

with the flux **j** given in (1). The is the so-called convective boundary condition we want to solve in this problem. The way we have set things up, the Zinc C matrix encodes the div $\mathbf{j}_k$ term in (1) while the Zinc f vector encodes the $\rho\mathbf{u}\cdot\nabla\omega_k$. Since C appears in the boundary condition (5), $\mathbf{n}.\mathbf{j}_k$=0 is the natural boundary condition we get.

There are two ways to solve non-linear problems in Zinc

1. Write a set of subroutines to encode the non-linearity, compile to a dynamic link library (DLL) and run with Zinc
2. Write expression in the Zinc input file encoding the non-linear behaviour

In this tutorial, we will show both methods and furthermore, we will prepare the DLL using both Fortran and C implementations. We will first consider the DLL method

DLL method

We will implement this method using the name cathodetoken. The input file, cathodetoken.zin, looks like this

```
key_sim=1
nvar=3
```

```
nreg=3
key_rdu=0
scale=1e-3

omega=0.1
nstep=30000
nstride=100
nstrideNL=1000

labels = x1 x2 p

region 1 elements 21 values C

! Row 1 of B
1 1 1 1 = $B11
1 2 1 2 = $B11
1 3 1 3 = $B11

1 1 2 1 = $B12
1 2 2 2 = $B12
1 3 2 3 = $B12

! Row 2 of B
2 1 1 1 = $B21
2 2 1 2 = $B21
2 3 1 3 = $B21

2 1 2 1 = $B22
2 2 2 2 = $B22
2 3 2 3 = $B22

! col 3, row 1 Q block
1 1 3 1 = $Q1
1 2 3 2 = $Q1
1 3 3 3 = $Q1

! col 3, row 2 Q block

2 1 3 1 = $Q2
2 2 3 2 = $Q2
2 3 3 3 = $Q2

! E section at bottom right
3 1 3 1 = $E
3 2 3 2 = $E
3 3 3 3 = $E

region 1 elements 2 values f
1 = $f1
2 = $f2

! ########################################################################

region 2 nodes 3 values
1 = x1c
2 = x2c
3 = pc

region 3 nodes 1 value
3 =pref
```

```
init
1 = x1c
2 = x2c
3 = (pc+pref)/2
```

The main concept here is *token replacement* whereby elements in the C and f matrices are defined, not by expression but by tokens beginning $. Any string can be used; when Zinc comes across the string it calls a user supplied function with the token name passed in as an argument. Thus, for the C functions, Zinc calls function cfun; for the f tokens, it calls ffun.

Here is the cfun function written in Fortran (cathodetoken.f90)

```
function Cfun(label,x,y,z,ur,dur,nvar,istep)

  use nlinc
  implicit none
  character label*(*)
  integer nvar,istep
  double precision Cfun,x,y,z,ur(nvar),dur(nvar,3)

  double precision D11,D12,D13,D21,D22,D23,D31,D32,D33
  double precision x1,x2,p,x3,w1,w2,w3,m,denom

  call NLinit

  x1=ur(1)
  x2=ur(2)
  p=ur(3)

  x3=1-x1-x2

  m=m1*x1+m2*x2+m3*x3

  w1=m1*x1/m
  w2=m2*x2/m
  w3=m3*x3/m

  denom=x1/(D12tilde*D13tilde)+x2/(D12tilde*D23tilde)+x3/(D13tilde*D23tilde)

  D11=((w2+w3)**2/(x1*D23tilde)+w2**2/(x2*D13tilde)+w3**2/(x3*D12tilde))/denom
  D22=((w1+w3)**2/(x2*D13tilde)+w1**2/(x1*D23tilde)+w3**2/(x3*D12tilde))/denom
  D33=((w1+w2)**2/(x3*D12tilde)+w1**2/(x1*D23tilde)+w2**2/(x2*D13tilde))/denom

  D12=(w1*(w2+w3)/(x1*D23tilde)+w2*(w1+w3)/(x2*D13tilde)-w3**2/(x3*D12tilde))/denom
  D13=(w1*(w2+w3)/(x1*D23tilde)+w3*(w1+w2)/(x3*D12tilde)-w2**2/(x2*D13tilde))/denom
  D23=(w2*(w1+w3)/(x2*D13tilde)+w3*(w1+w2)/(x3*D12tilde)-w1**2/(x1*D23tilde))/denom

  D21=D12
  D31=D13
  D32=D23

  if (label.eq.'$B11') then
     Cfun=p*m1*x1*(D11-D13)/RT

  else if (label.eq.'$B12') then
     Cfun=p*m1*x1*(D12-D13)/RT

  else if (label.eq.'$B21') then
     Cfun=p*m2*x2*(D21-D23)/RT

  else if (label.eq.'$B22') then
     Cfun=p*m2*x2*(D22-D23)/RT

  else if (label.eq.'$Q1') then
     Cfun=m1*x1/RT*(x1*(D11*(1-m1/m)-D13*(1-m3/m)) &
```

```
                  + x2*(D12*(1-m2/m)-D13*(1-m3/m)) &
                                    +D13*(1-m3/m))

  else if (label.eq.'$Q2') then
     Cfun=m2*x2/RT*(x1*(D21*(1-m1/m)-D23*(1-m3/m)) &
                  + x2*(D22*(1-m2/m)-D23*(1-m3/m)) &
                                    +D23*(1-m3/m))

  else if (label.eq.'$E') then
     Cfun=p*m*kappa/(RT*eta)
  else
     print *,'Error in Cfun: token not recognised:',trim(label)
     stop
  endif
```

end function Cfun

The operation of this function should be clear by considering (6)-(13). For example, if the "B11" token is called for, the function implements equation (9). This function calls Nlinit (see cathodetoken.f90) which simply reads in the cathodetoeken.con file in order to obtain the material properties stored there. This illustrates how user-written functions can call other functions and perform input/output operations as required.

The cfun function in C is written as

```
double cfun(char *label, double *x, double *y, double *z,
            double ur[3], double dur[3][3], int *nvar, int *istep, int length)
{

  double D11,D12,D13,D21,D22,D23,D31,D32,D33;
  double x1,x2,p,x3,w1,w2,w3,m,denom;

  NLinit();

  x1=ur[0];
  x2=ur[1];
  p=ur[2];

  x3=1-x1-x2;

  m=m1*x1+m2*x2+m3*x3;

  w1=m1*x1/m;
  w2=m2*x2/m;
  w3=m3*x3/m;

  denom=x1/(D12tilde*D13tilde)+x2/(D12tilde*D23tilde)+x3/(D13tilde*D23tilde);


D11=((w2+w3)*(w2+w3)/(x1*D23tilde)+w2*w2/(x2*D13tilde)+w3*w3/(x3*D12tilde))/denom;
D22=((w1+w3)*(w1+w3)/(x2*D13tilde)+w1*w1/(x1*D23tilde)+w3*w3/(x3*D12tilde))/denom;
D33=((w1+w2)*(w1+w2)/(x3*D12tilde)+w1*w1/(x1*D23tilde)+w2*w2/(x2*D13tilde))/denom;

  D12=(w1*(w2+w3)/(x1*D23tilde)+w2*(w1+w3)/(x2*D13tilde)w3*w3/(x3*D12tilde))/denom;
  D13=(w1*(w2+w3)/(x1*D23tilde)+w3*(w1+w2)/(x3*D12tilde)w2*w2/(x2*D13tilde))/denom;
  D23=(w2*(w1+w3)/(x2*D13tilde)+w3*(w1+w2)/(x3*D12tilde)w1*w1/(x1*D23tilde))/denom;

  D21=D12;
  D31=D13;
  D32=D23;

  if (equal(label,length,"$B11"))
    return p*m1*x1*(D11-D13)/RT;

  else if (equal(label,length,"$B12"))
```

```
    return p*m1*x1*(D12-D13)/RT;

  else if (equal(label,length,"$B21"))
    return p*m2*x2*(D21-D23)/RT;

  else if (equal(label,length,"$B22"))
    return p*m2*x2*(D22-D23)/RT;

  else if (equal(label,length,"$Q1"))
    return m1*x1/RT*(x1*(D11*(1-m1/m)-D13*(1-m3/m))
                    +x2*(D12*(1-m2/m)-D13*(1-m3/m))
                                    +D13*(1-m3/m));

  else if (equal(label,length,"$Q2"))
    return m2*x2/RT*(x1*(D21*(1-m1/m)-D23*(1-m3/m))
                   + x2*(D22*(1-m2/m)-D23*(1-m3/m))
                                     +D23*(1-m3/m));

  else if (equal(label,length,"$E"))
    return p*m*kappa/(RT*eta);
  else {
    printf("Error in Cfun: token not recognised\n");
    exit(1);
  }

}
```

Note that the token is passed in as a fixed length variable (length "`length`") rather than being null-terminated. For this reason we have written a function "equal" which the user should examine in the cathodetokenc.c file.

We used the free gcc compiler (including gfortran) to compile the Fortran and C DLLs.

```
gfortran -c –fnounderscoring cathodetoken.f90
gfortran -s -shared -mrtd -o cathodetoken.dll cathodetoken.o
```

and

```
gcc -c cathodetokenc.c
gcc -shared -mrtd -o cathodetokenc.dll cathodetokenc.o
```

The batch files compile.bat and ccompile.bat, included in the fuel cell directory, contain these commands.

Expressions method

Instead of entering literal constants for the C, f coefficients, it is possible to enter these as complex expressions. This approach is used in cathode.zin which does not require linking to a DLL file. Thus, the full expression for C matrix element C_{1111} is

```
region 1 elements 21 values C

! p*m1*x1*(D11-D13)/RT
1 1 1 1 = p*m1*x1/RT*((x2*m23+m3-m3*x1)^2/(x1*D23) + m2^2*x2/D13 + m3^2*(1-x1-x2)/D12   &
                     -(m1*(x2*m23+m3-m3*x1)/D23 + m3*(m1*x1+m2*x2)/D12 - m2^2*x2/D13))/ &
                     (m13*x1+m23*x2+m3)^2/(x1*fac1+x2*fac2+fac3)
```

This can be derived from (6) and (9) and using the D coefficient expressions just after Figure 1. Everything needs to be substituted in so that the C and f coefficients are written purely in terms of

1) The variables to be solved for x1, x2, p in this case (a labels command sets this up)
2) Constants in the cathode.con file. We have introduced a few new extra constants including fac1, fac2 and fac3 here (see cathode.con)

Writing out the expressions explicitly like this is ok, but it makes them more complicated to read. In the program based approach, the expressions were built up using intermediate variables which is not possible here. Also the DLL file method will work faster since the expressions are compiled into machine code. On the other hand, expressions like the one shown above need to be repeatedly interpreted by Zinc when it runs.

In summary, the programming/DLL method should be used for more complicated non-linearity systems and fuel cells certainly fit that category!

See the file cathode.zin for the full set of expressions.

Running the simulation

After a lengthy preamble, we are now ready to run the simulation. Fig 2 shows the simulation geometry which may be compared to Fig 1. The geometry is essentially 2-D so we have only two planes of elements in the third direction. Note that the distances in Fig 2 are in mm whereas the equations we have derived are in SI units (m). therefore a scale factor of 0.001 is needed and this is in the command

```
scale=1e-3
```

in the cathode.zin files. Finally let's look at the node fixing (Dirichlet boundary) part of the ZIN file and the initial condition

```
region 2 nodes 3 values
1 = x1c
2 = x2c
3 = pc

region 3 nodes 1 value
3 =pref

init
1 = x1c
2 = x2c
3 = (pc+pref)/2
```

This says that region 2 nodes will be completely fixed (see Fig 2) but region 3 nodes will be fixed only in terms of the pressure. The two mass fractions are allowed to vary.

The initialisation block (init) fixes the mass fractions and pressures at x1c, x2c and (pc+pref)/2. These constants are all in cathode.con.

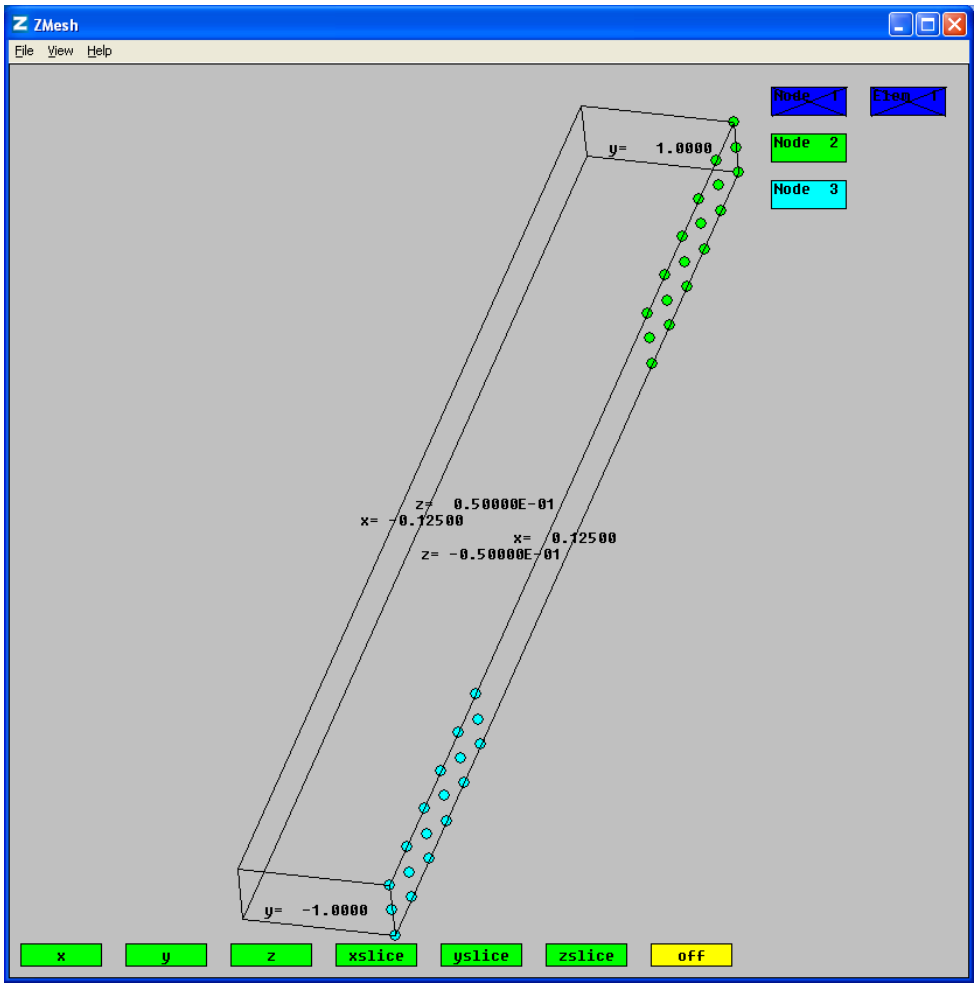


Fig 2: Fuel cell geometry. Mass fractions and pressures are fixed on region 2 nodes (green). Pressure only is fixed on region 3 nodes (cyan).

The Zinc window is shown in Fig 3. Note that Zinc has noticed the presence of the cathodetoken.f90 file and allows you to view it (you can change the source file extension). When you run zinc, the presence of the DLL file is detected and, in this case, cathodetoken.dll is copied into the Zinc directory. Zinc then runs linked to that DLL.

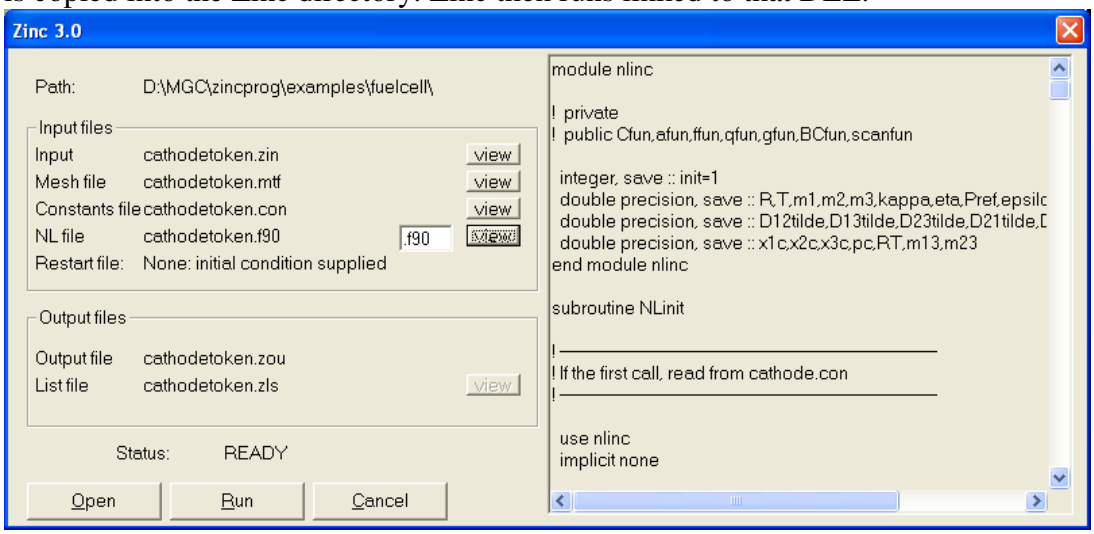


Fig 3: Zinc solver ready to run the fuel cell problem. Here the non-linear source code routines have the extension .f90 for a Fortran file. You can change this to match your language.

Having run the simulation, it is now time to post process with Zpp, Fig 4. Here we stick to simple output of the raw quantities: the mass fractions and the pressure. Fig 5 shows the simulation result viewed with Zmesh.

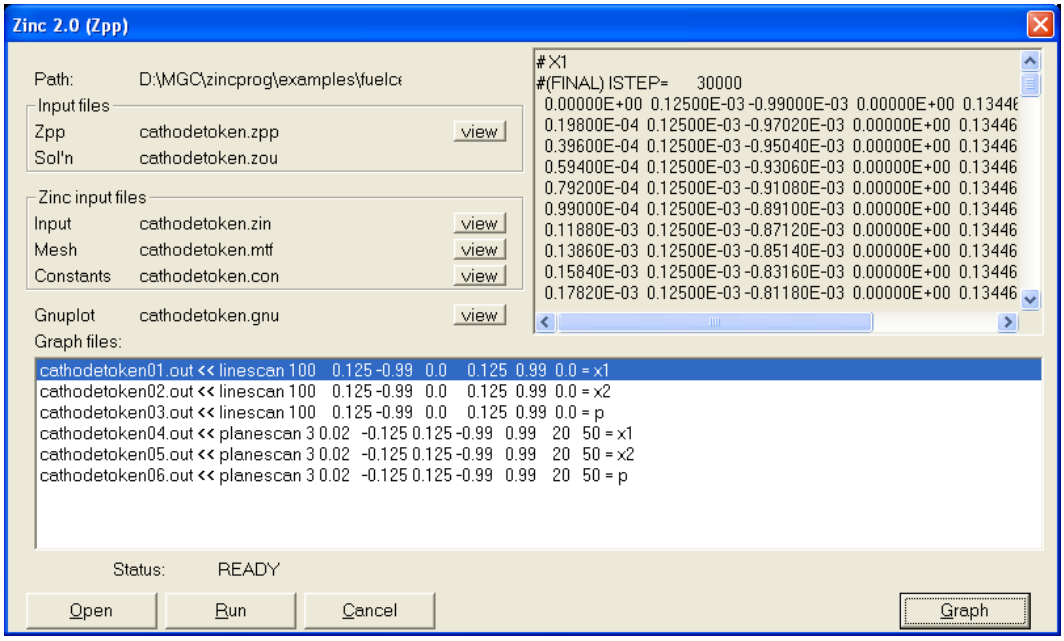


Fig 4: Post processing the fuel cell run

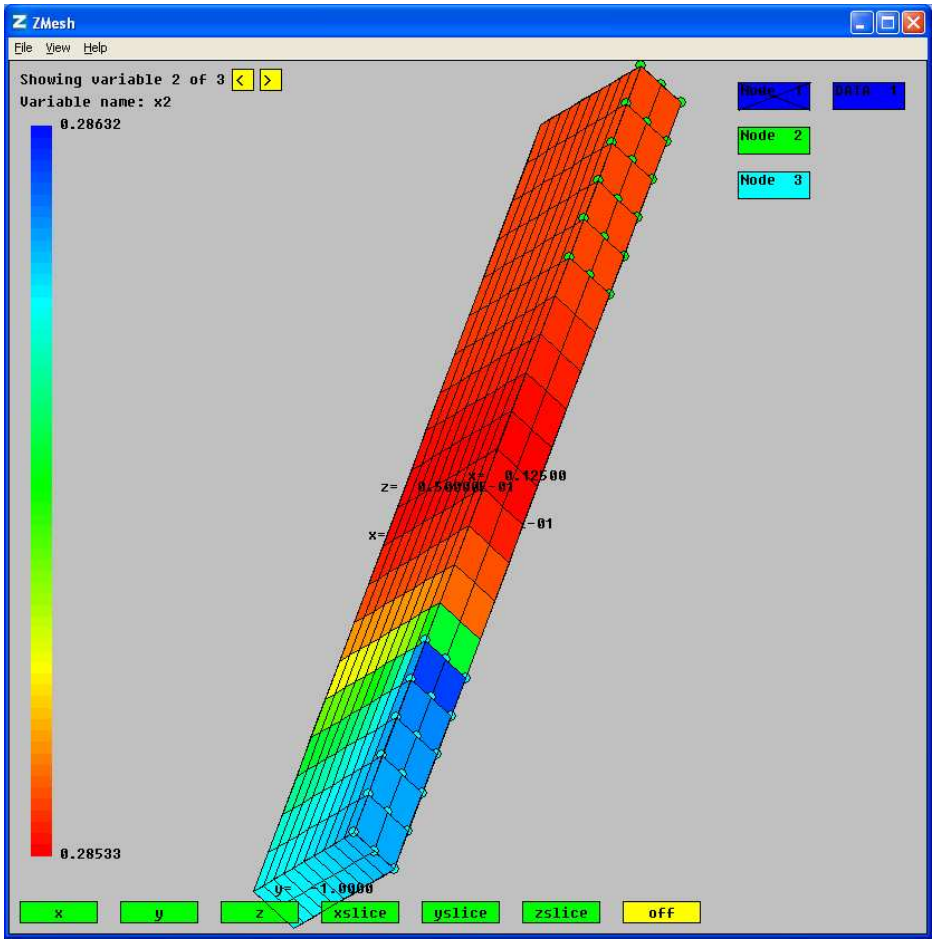


Fig 5: Fuel cell result viewed with Zmesh.

Results

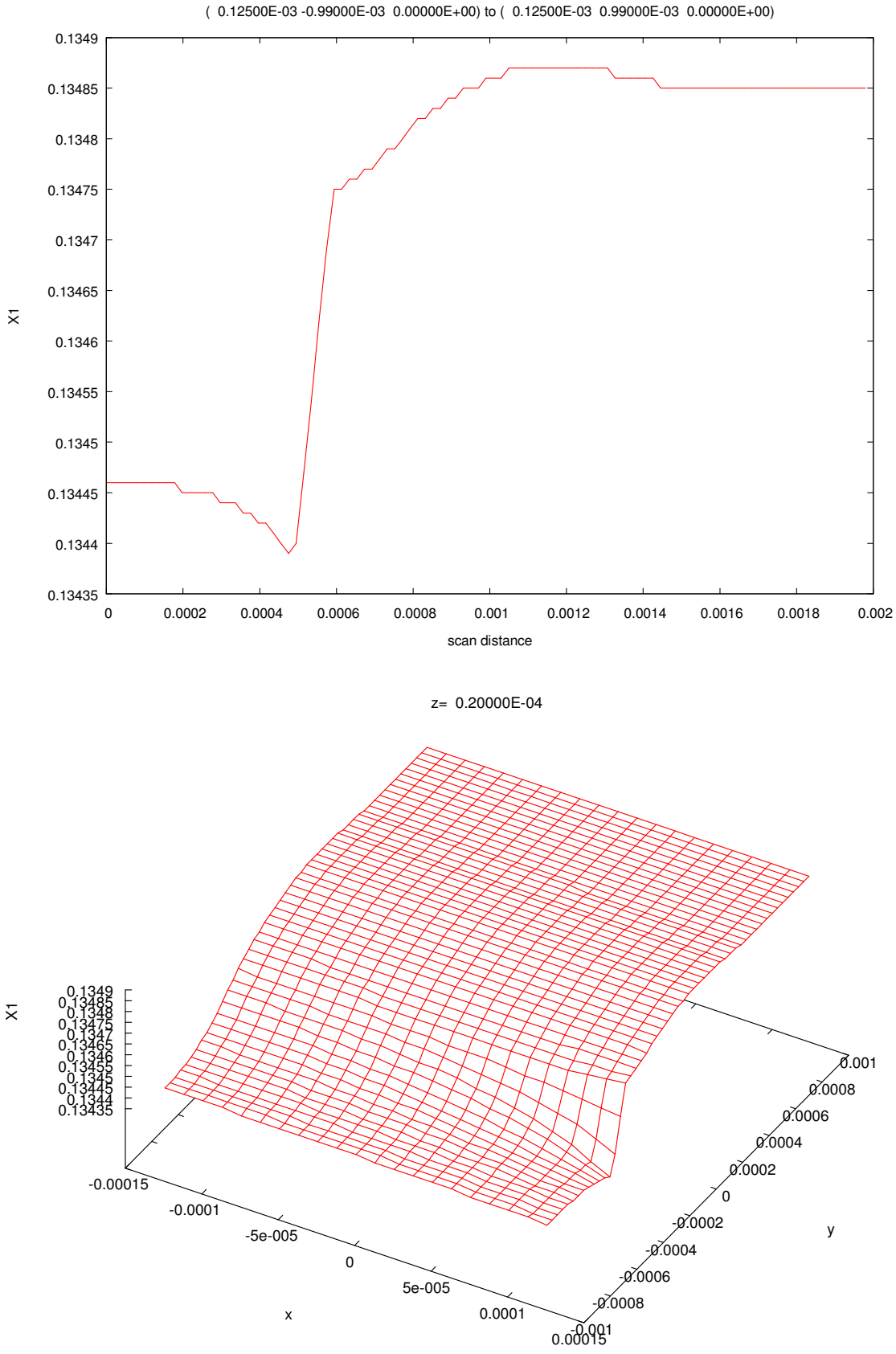


Fig 6: Variation of x_{O2}

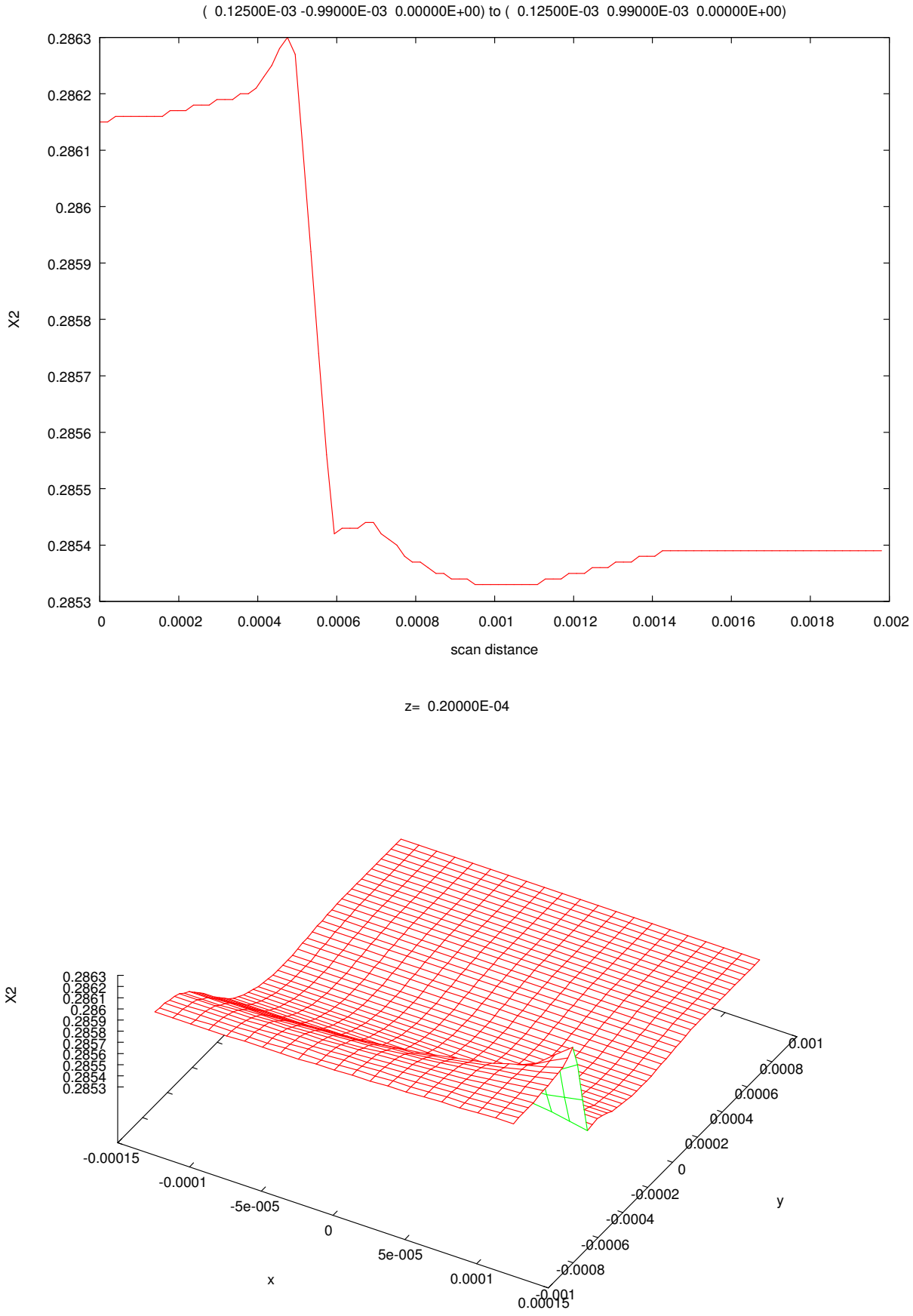
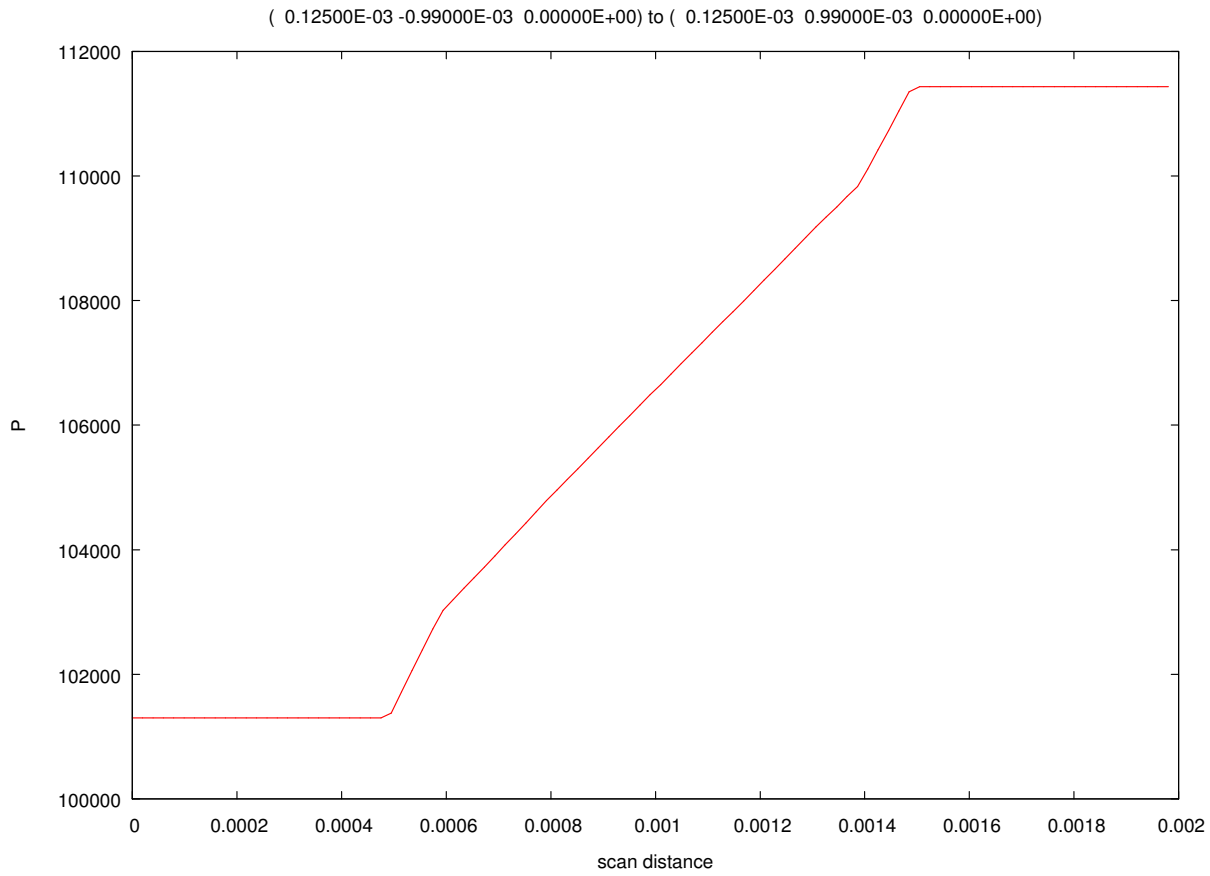


Fig 7: Variation of x_{H2O}



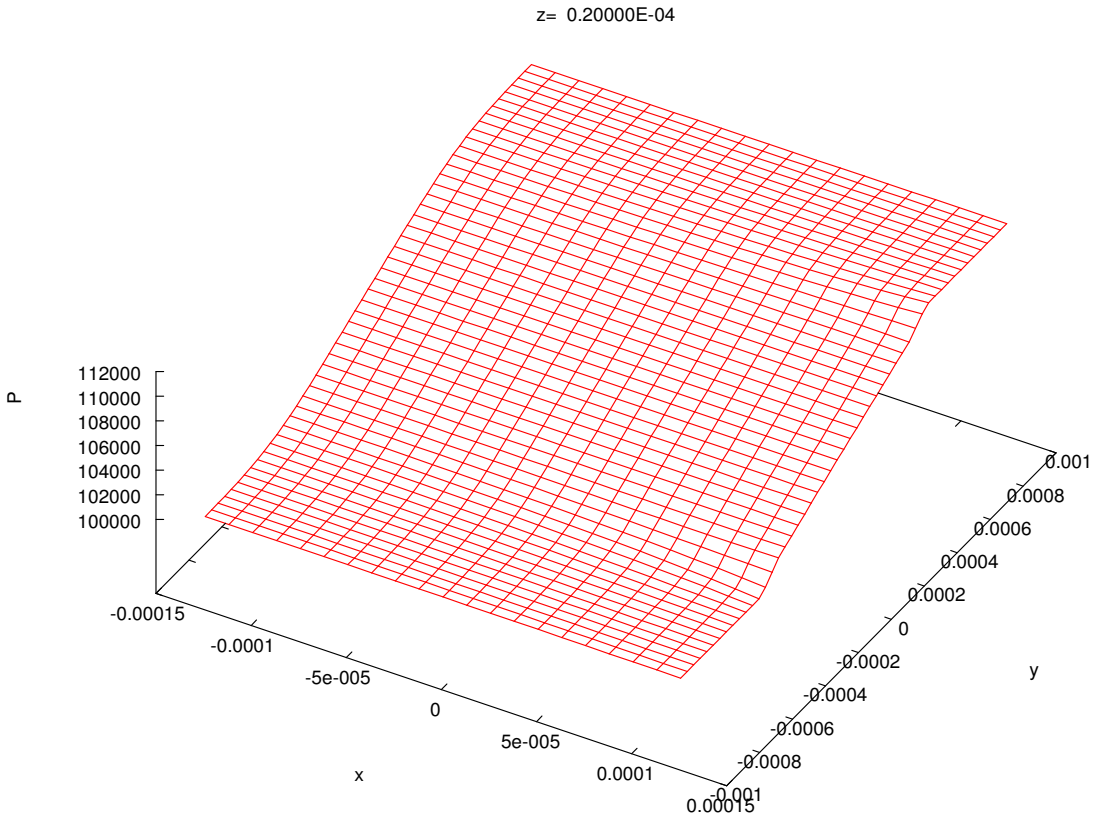


Fig 8: Variation of pressure

Figs 6-8 show the oxygen (x_1), water (x_2) and pressure (p) distribution in the sample.

Also in the directory

If the fuel cells runs take to long (especially the non-DLL version, cathode.zin) try uncommenting the lines

```
! Try this for a quicker result
! Nstep=10000
! Key_rdu=1
```

in the ZIN files of this directory. This will continue from a previous simulation rather than starting from scratch.

Appendix A: Description of differential equations

The differential equations solved by ZINC have the form

$$-\nabla.\left(\mathbf{C}\nabla\mathbf{U}\right)+\mathbf{a}\,\mathbf{U}=\mathbf{f}\,, \tag{A1}$$

which are written explicitly in component form as follows

$$-\left(\overline{C}_{ij}^{\alpha\beta}U_{,j}^{\beta}\right)_{,i}+a^{\alpha\beta}U^{\beta}=f^{\alpha}\,,\quad \alpha=1,\,2,\,....\,,\,N, \tag{A2}$$

where summation over 1, 2, …. N is implied by repeated *Greek superscripts*, and where summation over 1, 2, 3 is implied by *Arabic suffices*. Here, $f_{i,j}=\partial f_i/\partial x_j$. The structure of the (3N x 3N) **C** matrix when N = 2 is

$$\left[\begin{array}{ccc|ccc}
\overline{C}_{11}^{11} & \overline{C}_{12}^{11} & \overline{C}_{13}^{11} & \overline{C}_{11}^{12} & \overline{C}_{12}^{12} & \overline{C}_{13}^{12} \\
\overline{C}_{21}^{11} & \overline{C}_{22}^{11} & \overline{C}_{23}^{11} & \overline{C}_{21}^{12} & \overline{C}_{22}^{12} & \overline{C}_{23}^{12} \\
\overline{C}_{31}^{11} & \overline{C}_{32}^{11} & \overline{C}_{33}^{11} & \overline{C}_{31}^{12} & \overline{C}_{32}^{12} & \overline{C}_{33}^{12} \\
\hline
\overline{C}_{11}^{21} & \overline{C}_{12}^{21} & \overline{C}_{13}^{21} & \overline{C}_{11}^{22} & \overline{C}_{12}^{22} & \overline{C}_{13}^{22} \\
\overline{C}_{21}^{21} & \overline{C}_{22}^{21} & \overline{C}_{23}^{21} & \overline{C}_{21}^{22} & \overline{C}_{22}^{22} & \overline{C}_{23}^{22} \\
\overline{C}_{31}^{21} & \overline{C}_{32}^{21} & \overline{C}_{33}^{21} & \overline{C}_{31}^{22} & \overline{C}_{32}^{22} & \overline{C}_{33}^{22}
\end{array}\right]. \tag{A3}$$

Note that

$$\overline{C}_{ij}^{\alpha\beta}=\overline{C}_{\alpha i\beta j}$$

in the notation of the User Manual, Chapter 4. C components in the input ZIN file need to be entered in the order α, i, β, j.

The general form of the matrix $\bar{\mathbf{C}}$ is an (N x N) array of (3 x 3) matrices. The term $\nabla\mathbf{U}$ expands in component form as

$$\left(\nabla\mathbf{U}\right)^{\beta}_{j} = U^{\beta}_{,j}\,, \quad \beta = 1,\ 2,\ \ldots\ ,\ N, \quad j = 1,\ 2,\ 3\,. \tag{A4}$$

The term $\nabla\mathbf{U}$ is regarded as a (1 x 3N) column vector. When N = 2 it would be written

$$\nabla\mathbf{U} = \begin{bmatrix} U^{1}_{,1} \\ U^{1}_{,2} \\ U^{1}_{,3} \\ \hline U^{2}_{,1} \\ U^{2}_{,2} \\ U^{2}_{,3} \end{bmatrix}\,. \tag{A5}$$

The term $\mathbf{C}\nabla\mathbf{U}$ then expands in component form as follows

$$\left(\mathbf{C}\nabla\mathbf{U}\right)^{\alpha}_{i} = \bar{C}^{\alpha\beta}_{ij}\, U^{\beta}_{,j}, \quad \alpha = 1,\ 2,\ \ldots\ ,\ N, \quad i = 1,\ 2,\ 3\,. \tag{A6}$$

The term $\mathbf{C}\nabla\mathbf{U}$ is regarded as a (1 x 3N) column vector. When N = 2 it would be written

$$\mathbf{C}\nabla\mathbf{U} = \left[\begin{array}{ccc|ccc} \bar{C}^{11}_{11} & \bar{C}^{11}_{12} & \bar{C}^{11}_{13} & \bar{C}^{12}_{11} & \bar{C}^{12}_{12} & \bar{C}^{12}_{13} \\ \bar{C}^{11}_{21} & \bar{C}^{11}_{22} & \bar{C}^{11}_{23} & \bar{C}^{12}_{21} & \bar{C}^{12}_{22} & \bar{C}^{12}_{23} \\ \bar{C}^{11}_{31} & \bar{C}^{11}_{32} & \bar{C}^{11}_{33} & \bar{C}^{12}_{31} & \bar{C}^{12}_{32} & \bar{C}^{12}_{33} \\ \hline \bar{C}^{21}_{11} & \bar{C}^{21}_{12} & \bar{C}^{21}_{13} & \bar{C}^{22}_{11} & \bar{C}^{22}_{12} & \bar{C}^{22}_{13} \\ \bar{C}^{21}_{21} & \bar{C}^{21}_{22} & \bar{C}^{21}_{23} & \bar{C}^{22}_{21} & \bar{C}^{22}_{22} & \bar{C}^{22}_{23} \\ \bar{C}^{21}_{31} & \bar{C}^{21}_{32} & \bar{C}^{21}_{33} & \bar{C}^{22}_{31} & \bar{C}^{22}_{32} & \bar{C}^{22}_{33} \end{array}\right] \left[\begin{array}{c} U^{1}_{,1} \\ U^{1}_{,2} \\ U^{1}_{,3} \\ \hline U^{2}_{,1} \\ U^{2}_{,2} \\ U^{2}_{,3} \end{array}\right] = \left[\begin{array}{c} \bar{C}^{1\beta}_{1j} U^{\beta}_{,j} \\ \bar{C}^{1\beta}_{2j} U^{\beta}_{,j} \\ \bar{C}^{1\beta}_{3j} U^{\beta}_{,j} \\ \hline \bar{C}^{2\beta}_{1j} U^{\beta}_{,j} \\ \bar{C}^{2\beta}_{2j} U^{\beta}_{,j} \\ \bar{C}^{2\beta}_{3j} U^{\beta}_{,j} \end{array}\right]\,. \tag{A7}$$

It then follows that

$$\left(\,\nabla.\left(\mathbf{C}\nabla\mathbf{U}\right)\right)^{\alpha} = \left(\bar{C}^{\alpha\beta}_{ij}\, U^{\beta}_{,j}\right)_{,i}\,, \quad \alpha = 1,\ 2,\ \ldots\ ,\ N\,. \tag{A8}$$

The term $\nabla.\left(\mathbf{C}\nabla\mathbf{U}\right)$ is regarded as an (1 x N) column vector. When N = 2 it would be written

$$\nabla.\left(\mathbf{C}\nabla\mathbf{U}\right) = \begin{bmatrix} \left(\bar{C}^{1\beta}_{ij} U^{\beta}_{,j}\right)_{,i} \\ \left(\bar{C}^{2\beta}_{ij} U^{\beta}_{,j}\right)_{,i} \end{bmatrix}\,. \tag{A9}$$

The matrix $\mathbf{C}$ used by the ZINC code has components of the form $C_{\alpha i\beta j}$ which are given by

$$C_{\alpha i \beta j} = \bar{C}_{ij}^{\alpha\beta} \ , \qquad \alpha, \beta = 1, 2, ..., N, \ \ i, j = 1, 2, 3. \tag{A10}$$

Boundary conditions

In order that the differential equations (A2) have a unique solution a set of boundary conditions must be imposed on the external boundary of the system. Such boundary conditions are assumed to be of the form (generalised Neumann condition)

$$\mathbf{n} . \left(\mathbf{C} \nabla \mathbf{U} \right) + q \, \mathbf{U} = \mathbf{g} , \tag{A11}$$

where **n** is the outward unit vector on the external boundary, and where q is a prescribed scalar function and **g** is a prescribed vector function (dimension N). In component form this condition is written

$$n_i \bar{C}_{ij}^{\alpha\beta} U_{,j}^{\beta} + q \, U^{\alpha} = g^{\alpha} \ , \quad \alpha = 1, 2, ... , N. \tag{A12}$$